"Overmorrow"

Story of Procrastinators

A
Rajesh Kodaganti's
Procastination

OVERMORROW

. . .

Story of Procrastinators

By Rajesh Kodaganti

a novel of one software project, told along the lifecycle

Contents

Dedication

```
if (you.is(developer) || you.is(procrastinator)) {
    this.book.dedicatedTo(you);
} else if (you.have(deadline)) {
    this.book.dedicatedTo(you);
} else {
    // you will, eventually, qualify.
    this.book.dedicatedTo(you);
}

// tomorrow can wait.
// this page cannot.
```

PROLOGUE — THE WORD

> "The most expensive word in software is not 'production.' It is not 'urgent.' It is not even 'rewrite.' The most expensive word In software is 'tomorrow.' It costs nothing the day you say it. It costs everything the day it comes due. The day it comes due is, in my experience, never the tomorrow you thought it would be."
>
> — anonymous engineer, forum comment

There is, in old English, a word almost nobody uses anymore. The word is *overmorrow*. It means *the day after tomorrow*. You will not find it in most modern dictionaries. You will find it in the King James Bible, in the small honest mouths of people in the seventeenth century who, when they were postponing a thing, were specific about exactly how far they were postponing it.

We do not, in our century, talk that way. We say *tomorrow* when we mean *the day after tomorrow*. We say *the day after tomorrow* when we mean *next week*. We say *next week* when we mean *whenever this becomes a fire*. By the time the fire arrives, we have lost the words for the days, and we are standing in the smoke arguing about who said what at which meeting, with calendars in our hands that, if we are honest, say only one date, and the date is *too late*.

I am writing this book about a project I worked on. The project had a name. The name, on the kickoff slide, was *Project Overmorrow*. Somebody on the founding team — I have never been sure who — picked the name. I think it was a joke. I think the joke was supposed to be that the project was an ambitious one, and that *overmorrow* was a small poetic shrug about how every truly ambitious project has, in its bones, a date that keeps moving by another day, and another day, and another. I think the person who picked the name did not know that they were naming a tragedy. I think they thought they were naming a comedy. The two genres, viewed from inside the lifecycle of a software team, are distinguishable mostly by the timestamp on the postmortem.

I am a backend developer. I have been one for fifteen years. I joined the project I am about to describe as Developer 2, which is a thing I will not stop being, in some part of my own head, even after the events I am about to describe will have, formally, made me something else. I write code. I keep a notebook. I am, I have been told, careful. I am, I have also been told, quiet. The two adjectives are usually meant as compliments. They are not, in the world of a project that is on fire, always compliments. They are sometimes the names of two specific kinds of cowardice, and I had to learn this about myself the long way, which is to say, I had to learn it by being the cause of harm I could have prevented by opening my mouth.

This is the story of how I learned that. It is also the story of seven other people I worked with, who learned, in their own ways, in their own scenes, the same thing about themselves. None of them are villains. None of us were villains. We were a team of competent professionals who, for six months, *did not do the things we knew we should do*, and who told ourselves, every day, that we would do those things *tomorrow*. We were, all of us, procrastinators. Some of us procrastinated on tests. Some of us procrastinated on documentation. Some of us procrastinated on certificates, on contracts, on conversations, on the single Slack message that, sent in week three, would have prevented seventy percent of what came in week twenty-three. The shapes of our procrastinations were different. The shape of the *result* was the same. We built, by way of all our small daily deferrals, a thing that did not work, for people we had never asked, on a schedule we had agreed to without believing in, and we acted, the entire time we were doing it, like the next day was a country we would always be able to visit.

I used to think procrastination was a personality trait. I used to think it was the small private failing of a small private self. I do not think this anymore. I think procrastination is a *team disease*. I think it is the quiet assumption, made silently by every member of a group, that *somebody else* is going to write the doc / send the email / file the bug / have the conversation / fix the TODO. I think this assumption is shared, in nearly every team I have ever worked on, in nearly every company I have ever worked at, by approximately every person on the team, about approximately every important task. I think this is why the doc never gets written, the email never gets sent, the bug never gets filed, the conversation never gets had, the TODO sits in the codebase for four years and nine months until a new engineer reads it and thinks, *huh, strange*, and then leaves it there, because the new engineer also has things they would prefer to do tomorrow.

I am not going to tell you, in this book, how to fix this. I do not know how to fix this. I am not, on a good day, sure that I have fixed it in *myself*, let alone in a whole team, let alone in a whole industry. I am only going to tell you what happened to us. I am going to tell you who we were. I am going to tell you what we put off, and what it cost, and what eventually — slowly, embarrassingly, the long way — some of us learned to do instead. I am going to do it in small pieces, one scene at a time, organized along the phases of the software development lifecycle, because the lifecycle was the rope we hanged ourselves with and it seems only fair to use it as the structure for the confession.

A note on the form.

I will not use anyone's name. I will refer to my colleagues only by their roles on the team. The Tech Lead. The QA Tester. The Sales Manager. The Customer Support Rep. The Product Manager. Developer 1. Developer 3. The DevOps Engineer. The UX Designer. The UI Designer. The Security Engineer. The Director of Engineering. I do this for two reasons. The first is that the people deserve their privacy. The second, and more honest one, is that the *roles* are the point of the story. The story would have happened, in some shape, with any seven people in those roles, in any company, on any project of comparable size and ambition.

I have been on enough teams now to know that. The names are incidental. The procrastinations are not.

I will, at the start of every scene, give you a short anonymous quote. The quotes are not real. The quotes are, also, all real. They are the kind of small confession an engineer or a designer or a tester or a support rep posts on a forum at one in the morning, expecting nobody to read it, and then watching it accumulate, in twelve hours, a thousand replies that all begin with the words *us too*. They are the folklore of an industry that has been telling itself the same stories about its own behavior for as long as the industry has had deadlines. I have, in this book, treated those stories as the chorus. The team is the foreground. The chorus is the background. They are, on every page, in conversation with each other, because the specific is always in conversation with the general, and the general is what tells you that you are not alone.

I will refer to myself only as the Protagonist. I do this because I am, in this book, the most reliable narrator of my own behavior I have been able to manage, which is not the same as being a *good* narrator of my own behavior, which is in turn not the same as being the *hero*. I am not the hero. There is no hero. There are only people who, at one specific moment in the project, said *yes* to the thing they had been saying *tomorrow* about for too long, and who, in saying *yes*, allowed the team — slowly, unglamorously, in a way that surprised all of us — to begin to be the team it had agreed, in month one, to be.

If this book has a thesis, it is this. The thesis is not *do not procrastinate*. That is too easy. That is not true. Everyone procrastinates. The thesis is: *be honest, out loud, with the team, about the things you are procrastinating on, before they become the things that break the project*. Honesty is, in my experience, the only intervention that works. It does not work because honesty is a virtue. It works because honesty, said early, lets *someone else* on the team — a colleague, a tester, a support rep, a manager — do the thing you have been unable to do alone. Procrastination, in a team, is almost always a problem of *secrecy*. It is not the deferral that kills the project. It is the *not telling anyone you are deferring*. The fix is not willpower. The fix is disclosure.

I will give you, in the end, the smallest possible version of this fix. It is one word. The word is *today*. We will get to it. We will get to it the long way. We will get to it the only way that, on this particular team, it could have been gotten to, which is the wrong way, the expensive way, the way that involved most of us discovering, in front of each other, the specific shape of our own avoidance.

But that is the end of the book.

For now, on a Monday in month one, in a glass-walled conference room called the Innovation Suite, we are about to have our kickoff. The Product Manager is joining remotely. He is, the calendar invite says, *between meetings*. He will be between meetings for the next six months. We do not yet know this. Hope, at this stage, is a renewable resource.

It will not, at this stage, occur to any of us that the six months are already, in some quiet way, beginning to go wrong. We are, all of us, full of energy. We are, all of us, full of plans. We are, all of us, full of *tomorrow*.

We have plenty of it.

We will, in the end, run out.

— END OF PROLOGUE —

Chapter 1 — Planning the Feature

1.1 — UX

> "We did six weeks of user interviews. The findings live in a Miro board nobody can find anymore. The person who ran it left in March. The board may or may not be in her personal account. We have, essentially, vibes."
>
> — anonymous designer, online community

The kickoff was on a Monday, which should have been the first warning. Nothing important has ever started well on a Monday at ten in the morning, and yet here we were, eight people and one empty chair, gathered in the glass-walled room the building manager had named, with corporate optimism, the Innovation Suite.

The Innovation Suite had a smart screen that did not work and a plant that was, depending on the angle of the light, either thriving or extinct. The empty chair belonged to the Product Manager, who was, the calendar invite said, "joining remotely." The Product Manager would be joining remotely for the next six months. We did not yet know this. Hope, at this stage, was a renewable resource.

The UX Designer stood at the front. She had a stack of printouts and the energy of someone who had stayed up too late reading about Jobs To Be Done frameworks. On the screen behind her, a slide read:

PROJECT OVERMORROW The Unified Customer Portal Discovery & Vision

"So," she said, sliding her printouts across the table, "I've been talking to users."

This was true, in the way that "I've been meaning to call my mother" is true. She had, in fact, talked to users. Twelve of them. Over the course of two months. The interviews, she explained, were rich. Insights had been gathered. Patterns had emerged. A persona had been born — a composite figure she had named, on a sticky note, "Frustrated Fiona."

"Where can we read the findings?" the QA Tester asked, because the QA Tester is constitutionally incapable of vibes.

"They're in my notebook," the UX Designer said. "I'll write them up this week. Latest, next week. I just want to do one more synthesis pass."

I wrote, in my private notes, *user research → not yet documented*, and underlined it twice, because some part of me already knew.

The Sales Manager leaned forward. He had the bright, hungry look of a man who was about to commit other people to things. "Who's Fiona, exactly? Like, what does she want?"

"Fiona," the UX Designer said, with the tone of a sommelier introducing a wine, "wants to feel *in control*."

"Of what."

"Of her data."

"Right, but specifically —"

"Of all of it."

The Sales Manager nodded slowly, the way you nod when someone has said something profound that you do not understand and do not want to admit you do not understand. He wrote *FIONA: WANTS CONTROL* on his notepad, in capital letters, and circled it. This was, I would learn later, the most enduring artifact of the discovery phase. He would refer to it in three separate sales calls. Two clients would ask who Fiona was. He would say, "She's a representative user," and the clients would nod the same slow nod, and somehow, mysteriously, the contracts would still get signed.

The UX Designer continued. There would be a research repo. There would be a journey map. There would be a service blueprint — "That's the one with the swim lanes," she said, in case any of us had forgotten what a service blueprint was. (We had. All of us. We always do.) These would be ready, she said, by end of sprint two.

"Sprint two," the Tech Lead said. "Cool. And in the meantime?"

"In the meantime," she said, "trust the vision."

The Product Manager, from the screen, gave a thumbs-up. He was, we could see in the small window, in a café. There was a croissant on the table. Behind him, a barista was foaming milk with what looked like real artistry. The Product Manager said, "I love this energy," and his audio cut out.

The Frontend Developer — Developer 1, who I would later watch build an entire login screen against a fake API because the real one didn't exist yet — leaned back in her chair and stretched. "I mean, the vision is clear, right? Customer logs in, sees their stuff, exports their stuff, feels in control. Like Fiona."

"Right," said the UX Designer.

"Right," said the Sales Manager.

"Right," said the Tech Lead, with the slightly distant tone of someone already drafting a system diagram in his head and absolutely not listening to anyone else.

I did not say "right." I wrote, instead: *what does "their stuff" mean*. Then: *what is the actual user journey from login to export*. Then: *what error states exist*. Then: *what happens when a user has no data yet*. Then: *what happens when a user has too much data*.

Then I wrote, at the bottom of the page, the line I would re-read many months later, in the dark, with my forehead against my desk:

We are about to build a thing for a person we have not yet finished describing.

The UX Designer wrapped up. There was applause, of the polite clapping-with-three-fingers variety. The Product Manager's video came back. He said, "Sorry, what did I miss," and the Sales Manager said, "Vision," and the Product Manager said, "Love it," and his audio cut out again.

I stayed behind to help stack the chairs. The UX Designer was gathering up her printouts. They were, I noticed now, mostly blank.

"Hey," she said, "do you want me to send you the persona doc when I write it up?"

"Yes," I said. "Please."

"I'll do it tomorrow," she said, and smiled.

She would not do it tomorrow.

She would do it, eventually, the morning of the beta launch, under a different filename, in a panic, because the Sales Manager needed it for a slide. By then, of course, we had built the wrong thing for a slightly different person, and "Fiona," like all the best fictions, had quietly become whoever we needed her to be.

But that was tomorrow. And tomorrow, on that Monday in the Innovation Suite, was a country none of us thought we would ever have to visit.

— END OF 1.1 —

1.2 — UI

> "The Figma had three thousand frames. Beautiful ones. Award-winning ones. None of them said which screen came after which. The dev team called it 'the museum.'"
>
> — anonymous product designer, design forum

The UI Designer's review was on Wednesday, two days after the kickoff. He scheduled it for ninety minutes, and we let him, because no one had the heart to say no.

He arrived with his laptop already open to Figma, the way hunters arrive at a forest already with a rifle. He had been working, he said, on the design system. The design system was going to be "comprehensive." It was going to be "scalable." It was going to be, and here he paused for emphasis, "*tokens- first.*"

The Sales Manager nodded the slow nod again. He wrote *TOKENS- FIRST* on his notepad. He did not circle it this time, which was, on reflection, restraint.

"Show us what you've got," the Tech Lead said.

The UI Designer turned his screen. We saw it then for the first time: the museum. Hundreds of frames, neatly arranged in a grid, each one a polished, beautiful, perfectly aligned screen. There was a login screen with a soft gradient that suggested a sunrise over a lake. There was an empty state with an illustration of a small fox holding a clipboard. There was a settings page so elegant I briefly forgot what we were doing.

There were, I noticed, no arrows between any of the frames.

"This is gorgeous," Developer 1 said, and she meant it. Her face had the look devs sometimes get when they realize the design is going to make them look like geniuses, which is the same face they get when the design is going to make them weep.

"Thanks," the UI Designer said. "I've been iterating."

"Question," I said. "Where does the user go after they hit 'Export'?"

A small silence.

"Like — do they get a confirmation modal? Do they see the export in a list? Does an email arrive? Where do they *land*?"

The UI Designer scrolled through his frames, slowly, with the contained anxiety of a man who knows the answer is in here somewhere and also knows it might not be.

"I haven't done that flow yet," he said, finally. "I wanted to nail the entry points first."

"Okay," I said. "And the empty state, when a user has zero records — does that have its own screen, or do we just hide the table?"

"Good question. I'll add it to the list."

He had a list. He always had a list. The list, by the end of the project, would have one hundred and forty-seven items on it, of which approximately eleven would ever get checked off. We did not know this yet. We were, instead, looking at the sunrise over the lake.

The UX Designer, who had attended virtually, unmuted herself. "This is so aligned with the vision," she said. "I love the fox."

"Thanks," the UI Designer said. "I drew it last weekend. I was going to use a stock illustration but it just didn't feel *us.*"

The Tech Lead, who had been silent for an unusual length of time, spoke. "When can the engineers start building against this?"

"Well," the UI Designer said, "the components aren't in the library yet. I'm using local components. I want to do a pass on the spacing system before I export them."

"How long is a pass."

"Honestly? End of next week. Latest, the week after."

"And in the meantime, the engineers..."

"They can work off the screens," the UI Designer said brightly. "Just, you know, eyeball the spacing."

I watched Developer 1's face. Developer 1 had, in her career, "eyeballed the spacing" perhaps three thousand times. Each time had aged her by a year. She nodded, with the cheerful desperation of someone agreeing to something she would later have to undo.

"What about the design tokens," I asked. "Are those in code yet?"

"They're in Figma."

"Right, but in code. So we can use them in the app."

"I'll export them. I just want to do one more pass on the contrast ratios."

The contrast ratios. The contrast ratios he would still be "doing one more pass on" eleven weeks later, by which time Developer 1 would have hard-coded the entire color palette as hex values in a file called `colors.ts` that she would, in her own words, "burn down later."

There was a thread on a developer forum — I'd read it once, years ago, the way you read horoscopes for a sign that isn't yours — about a team that had launched a product with three slightly different shades of blue across their app, because three different developers had each eyeballed the spec differently. The post had over a thousand comments. The most upvoted reply was a single sentence: *we are still finding new blues to this day.* I thought of that thread now. I did not say anything.

"I love this," the Sales Manager said. "Can I have these screens for the deck?"

"They're not finished," the UI Designer said.

"They look finished."

"They're not. The export flow, the empty states, the error states, the loading states, the responsive breakpoints —"

"They look finished, though," the Sales Manager said again, and he said it the way a man says *I'm going to take these whether you say yes or no*, and we all knew, in that moment, that screens which were not finished were about to be shown to clients as though they were finished, and that those clients were going to sign contracts based on them, and that the contracts would contain the word "as demonstrated" in a clause we would all, later, be forced to read very carefully.

The UI Designer hesitated. Then he sighed, the small sigh of the perpetually overruled, and he said, "Fine. But put a 'concept' watermark on them."

The Sales Manager said, "Of course," in the same voice the wolf uses when the grandmother asks about the teeth.

The meeting wrapped. The UI Designer began the long, beautiful work of doing one more pass. The Sales Manager began the long, ugly work of removing the "concept" watermark in PowerPoint. Developer 1 opened a new file in her editor and typed, at the top of it, in a comment she would never delete:

// TODO: replace with real design tokens when they ship

The TODO would still be there, unchanged, the day she resigned. But that was, of course, much later.

That afternoon, walking back to my desk, I thought about Fiona. Fiona who wanted to feel in control. Fiona who, somewhere, was trying to export something. Fiona who, in the version of the product we were about to build, would click "Export" and land in a screen that did not, technically, exist yet.

I added a line to my notes:

the screens are beautiful. the journeys are missing. we are building a museum, not a route.

Then I closed my notebook, because it was four in the afternoon, and the espresso machine was free, and I told myself I would think about it tomorrow.

I would. Just not in the way I imagined.

— END OF 1.2 —

1.3 — SOFTWARE ARCHITECTURE

"There was an architecture document. It was a single Confluence page. Its only content was the word 'TODO' and a Visio diagram of three boxes with an arrow that didn't connect to anything. It had been edited, last, in 2019. The author had left the company in 2020. It was 2024."

— anonymous engineer, internal wiki rant

The architecture session was on the Friday afternoon of the first week, which meant that everyone arrived in the Innovation Suite with the specific glaze of people who had stopped processing new information at lunchtime.

The Tech Lead had a marker. The Tech Lead always had a marker. He was, fundamentally, a marker-shaped human being.

"Okay," he said, "let's t-shirt size this thing. Don't overthink it."

"Don't overthink it" is, in software, the four-word incantation that summons the demon. I had heard it before. We all had. We all said *of course, of course*, and we all settled in to underthink it together.

He drew, on the whiteboard, three boxes.

[Frontend] → [API] → [Database]

He stepped back. He looked at it the way a sculptor looks at a finished piece. "There. That's the system."

"What about authentication," I said.

"Right. Yes." He drew a fourth box, off to the side, and labeled it *Auth*. He did not connect it to anything.

"And the streaming layer," I said. "For the realtime widgets."

"Streaming, yeah. Big topic." He drew a fifth box. He labeled it *Streaming?*. The question mark was, in retrospect, the most honest mark made on the board that day.

"And the export pipeline."

"Export, yes. We talked about this. Async. Probably a queue. We'll figure it out."

The board now had eight boxes. Three of them had question marks. Two of them had no arrows. One of them, I was pretty sure, had been there from a previous meeting and was actually about a different project entirely.

"So," the Tech Lead said. "Sizes. Frontend?"

"Medium," said Developer 1, who had not actually been listening and was, instead, writing a Slack message to her partner about dinner.

"API?"

"Medium," I said, "if we use the legacy framework. Large if we modernize."

"We're using the legacy framework," he said.

"Are we sure? Because the legacy framework doesn't support streaming, and the streaming question-mark box —"

"We're using the legacy framework. We can wrap it for streaming. It'll be fine."

I wrote, in my private notes: *we are wrapping a non-streaming framework to do streaming. this is the kind of decision we will be paying for in November.* I did not say it out loud. I told myself that was being polite. It was not being polite. It was being avoidant. There is a difference, and the difference is exactly six months long.

"Database?"

"Small," he said, before anyone could answer. "It's just CRUD."

"It's not just CRUD," the QA Tester said. "There's the reporting query. The dashboard joins six tables."

"Right, but that's a polish-phase thing. Get it working first, optimize later."

"There's never a polish phase."

"There's always a polish phase."

She did not argue. She wrote it down. She had, by this point in her career, written down many such promises. She had a notebook full of them. She referred to it, privately, as *The Book of Mañana.*

"DevOps and infra?"

The DevOps Engineer had been silent the entire meeting. He was holding a cup of cold coffee with both hands. He looked at the board. He counted the boxes. He counted the question marks. He took a long, slow sip of the coffee. Then he said, with the quiet finality of a man laying flowers on a grave:

"Small."

"Great," the Tech Lead said. He drew a circle around all the boxes and wrote *6 MONTHS* underneath it. It looked like a wanted poster.

"Cool," said the Sales Manager, who had appeared in the doorway at some point and was now standing there with the air of a man about to commit several professional crimes. "Cool cool cool. So I can tell the clients October."

"You can tell the clients October," the Tech Lead said.

I raised a hand. Halfway. The way you raise a hand when you're not sure you want to be called on. "Should we — write any of this down? Like, an actual architecture doc?"

The Tech Lead looked at me. He looked at the whiteboard. He looked at the Sales Manager, who was already on his phone, typing a message that I knew, with absolute certainty, contained the word *October*.

"Yeah," he said. "Good idea. Could you draft it?"

"Sure," I said.

"No rush," he said. "Just the high-level. Whenever you have a free moment."

No rush. I would think about those two words for a long time. *No rush* is the second most dangerous phrase in software. The first is *we'll fix it later*. The third is *don't overthink it*. We had, in a single seventy-five-minute meeting, managed to deploy all three.

That night, at home, I opened a new file in my editor. I named it `architecture-notes.md`. I typed, at the top:

Project Overmorrow — Architecture # Author: me # Status: DRAFT # Last updated: today

I sat with my fingers above the keyboard. I thought about the streaming wrapper. I thought about the auth box that didn't connect to anything. I thought about the six-table join. I thought about Fiona.

I thought about a story I had read once, on one of those late- night tabs you open and never close, about an engineer who had inherited a system with no documentation. He had spent his first three months on the job doing nothing but reading source code. His final exit interview, two years later, had contained a single line of feedback: *the only architecture doc is the production incident.* The post had been shared a quarter of a million times. People had laughed. People had also, I noticed in the comments, mostly been crying.

I started typing. I got as far as:

1. Overview The Customer Portal is composed of:

And then I stopped. The phone buzzed. It was the Sales Manager, in the team channel:

SALES MANAGER: hey just confirmed with client #1 — they want a demo of login + dashboard in 8 weeks. just the happy path. doable?

I watched the typing indicator from the Tech Lead.

TECH LEAD: should be fine 👍

I closed the laptop. I told myself I would finish the doc tomorrow. I went to bed. I lay in the dark, listening to the hum of the refrigerator, and I knew — in the precise, exhausted way you know things at midnight — that I was not going to finish it tomorrow. I was going to put it off. I was going to become, in my own quiet, well-organized way, exactly the problem I was about to spend six months blaming on everyone else.

I closed my eyes. I thought, with the small, slippery comfort of self-deception:

Tomorrow. Definitely tomorrow.

The refrigerator hummed on.

— END OF 1.3 —

1.4 — USER ACCEPTANCE TESTS

"The UAT plan was a single email from the PM to the client that said 'looking forward to your feedback!' The feedback came as a seventeen-page PDF the day before launch. I read it in the parking lot. I cried in the parking lot. I am still in the parking lot."
— anonymous PM, Slack screenshot

The UAT planning meeting was set for the second Tuesday. I remember this because the calendar invite came in twice — once from the Product Manager, once from the Sales Manager — and the two invites had different times, different agendas, and one of them had the wrong client on the attendee list.

I joined the call at the time on the first invite. I was alone in the meeting for nine minutes. I sat there, on camera, in my home office, watching myself watch myself. I thought, *this is how it begins. This is the year. I am being slowly erased by my own organization.* Then the rest of them filed in, late, distracted, eating various lunches.

The Product Manager joined last. He was driving. He should not have been driving. His face moved across the screen in a way that suggested at least one freeway exit. The wind was loud.

"Hey team," he said. "Sorry, between meetings. Can someone catch me up?"

The Sales Manager took this as his opening. "So! UAT! User Acceptance Testing! For the people who don't know — that's where we get the actual customers to test the product before we launch it, to make sure it does what they need."

He said this for the benefit of, presumably, the Product Manager, who did, in fact, need this explained to him. The Product Manager nodded. The freeway exit happened. We waited.

"Right," said the Product Manager, eventually. "UAT. Love it. What's the plan."

There was a small silence. The kind of silence where everyone is quietly hoping someone else has the plan.

"I thought you had the plan," the QA Tester said.

"I thought *you* had the plan," the Product Manager said.

"UAT is not a QA function," she said, in the tone of a person who has explained this exactly forty-seven times to exactly forty-seven different people.

The Sales Manager jumped in. "We'll co-design it with the clients. They'll tell us what they want to test. It'll be very collaborative."

"What does that mean, *operationally*," I said.

"It means," the Sales Manager said, "that I'll set up a kickoff call with each of the three pilot clients, and we'll walk them through the product, and they'll tell us what scenarios matter to them."

"When."

"Soon. Before the beta. Definitely."

"When is the beta," I said.

"Month four," the Tech Lead said.

"And when is *soon*," I said.

The Sales Manager smiled. The smile of a man who had not, in fact, scheduled anything yet, and who was going to spend the next four months not scheduling it. "I'll send the invites this week."

I wrote, in my notes: *UAT plan — none. Client kickoff calls — none scheduled. Acceptance criteria — undefined. Sign-off process — undefined. Definition of "done" — vibes.* Then I underlined *vibes* three times, because I had recently gotten a new pen and was enjoying it.

The QA Tester took a breath. "Okay. Can we at least agree on what we're going to ask the clients to *do* during UAT? Like, written test scenarios? A checklist? Anything?"

"Sure," the Product Manager said. "I'll draft that."

"By when."

"Soon."

"That's not — okay. Put a date on it. Any date. Throw a dart."

He pulled over. We all heard it. The hazard lights clicked on, audibly. The Product Manager was now stationary, on the shoulder of a freeway, fully present for the first time in two weeks. He looked into the camera with the haunted eyes of a man who had not, in his entire career, been asked to throw a dart.

"End of next week," he said.

"Thank you," the QA Tester said. She wrote it down. She underlined it. We all knew, in our hearts, that he would not do it by end of next week. We did not say so. We had, all of us, elected to be polite about this fact, which is to say, we had elected to be complicit in it.

"Great," the Sales Manager said. "And from my side — I'll get client kickoffs scheduled by, let's say, end of the month."

Developer 1 raised her hand. She had been very quiet. "Just asking — when the clients give us feedback during UAT, who's going to triage it? Like, sort the must-fix from the nice-to-haves?"

"I'll do it," the Product Manager said.

"While driving," I said. I said it before I could stop myself. It came out small and dry.

There was a beat. The Tech Lead snorted into his coffee. The Sales Manager pretended to be very interested in his laptop. The Product Manager laughed, generously. "Fair," he said. "Fair. I'll do it from a desk. With both hands."

I felt, briefly, like a person.

The meeting ended without an agenda for the next meeting, which is the corporate equivalent of leaving the dinner table without clearing your plate. We all logged off. I sat there, in my home office, looking at the blank Zoom screen.

I had read, once, on a developer forum, the story of a team that had skipped UAT entirely. They had told themselves they'd do it "in production" — that real users hitting the real product was the *truest* form of acceptance testing. The launch had gone well, technically. The product had not gone well, practically. The first three customers had used the system in ways the team had never imagined, none of them charitably. The top-rated comment on the post had been: *we built a product that worked. they wanted a product that helped.*

I closed my laptop. I opened my notebook. I wrote, in big block letters across a fresh page:

UAT IS NOT A FORMALITY. UAT IS THE LAST CHANCE TO LEARN WE BUILT THE WRONG THING.

Then, underneath, smaller:

nobody is going to organize this. nobody is going to organize this. nobody is going to organize this.

I read it three times. I closed the notebook. I told myself that, technically, I was not the PM. That, technically, this was not my problem. That, technically, the right thing to do was to focus on my code and trust the process.

I would think about that word, *technically*, many times in the months to come. I would think about it the way a man thinks about the small, polite lie he told at the start of a long fall.

For now, though, I went to make coffee. The kettle clicked on. Somewhere on a freeway, the Product Manager was almost certainly driving again. Somewhere in a Figma file, the UI Designer was doing one more pass on the contrast ratios. Somewhere in the Sales Manager's calendar, three client meetings remained unscheduled.

And somewhere, in a country none of us had visited yet, a woman named Fiona was waiting to be in control of her data.

She would have to wait a little longer.

— END OF CHAPTER 1 —

Chapter 2 — Developing the Feature

2.1 — FRONTEND DEVELOPMENT

"I built the entire dashboard against a hardcoded JSON file named `pretend.json`. The day before launch, the backend was finally ready. The shapes did not match. They had never matched. Nobody had thought to check."
— anonymous frontend developer, conference talk

Developer 1 was, by any reasonable measure, the most productive person on the team. She produced commits the way other people produce breath. She produced them in the morning, at lunch, in the evening, on the train, on weekends, on holidays, and once, memorably, during her own birthday dinner.

She did not, however, produce a *working* product. That was a separate question, and one we had all, in our quiet way, agreed not to ask.

What she produced was a beautiful frontend. It was responsive. It was animated. It had skeleton loaders that breathed gently, in and out, like sleeping cats. It had toast notifications that slid in from the corners with the precise easing curve of a luxury car door closing. It used the design tokens — well, it used hex codes that approximated the design tokens. The design tokens themselves, of course, were still in Figma, where the UI Designer was doing one more pass on the contrast ratios.

The frontend talked to a file called `pretend.json`. It had been, originally, called `mock-data.json`, but Developer 1 had renamed it after one too many bugs were filed against it as though it were the real thing. *pretend.json* made the situation clearer.

`pretend.json` contained a beautifully shaped imaginary universe. It had three users. It had twelve transactions. It had two billing periods. It had, helpfully, an "edge case" record where every field was the maximum length, so that Developer 1 could check the layout did not break.

The actual API, when it eventually existed — and there was, in the second sprint, a brief fascinating period when the API did not so much *exist* as *intermittently happen* — would not return shapes that matched `pretend.json`. The actual API would return shapes that matched the database schema, which had been designed in 2019 by someone who no longer worked at the company and who, judging by the column names, had been operating on a very personal definition of the word "user."

But that was a problem for later. *Later* was Developer 1's favorite tense.

I went to her desk on a Thursday in the third week to ask her about something.

"Hey," I said, "quick one. The dashboard widget — when it has no data, what does it render?"

"Empty state," she said, without looking up.

"Right, but what *empty state*. Like, where does that component come from?"

She glanced at her screen. She was deep in the middle of a CSS animation that involved, as far as I could tell, three separate keyframe sequences and a cubic-bezier curve she had derived from first principles.

"I'll wire that up later," she said.

"What's it rendering now?"

"Nothing."

"Like, blank?"

"Like, the layout collapses and the widget has zero height."

"Doesn't that look kind of broken?"

"It does," she said cheerfully, "if there's no data. But there's *always* data in `pretend.json`. So in dev, it looks fine."

I stood there. I tried, with great effort, to remain a calm, collegial backend engineer who minded his own service boundary.

"Developer 1," I said, "in production, there will be users with no data."

"There will," she agreed.

"And the layout will collapse."

"It will."

"And we will get bug reports."

"We will," she said, and she did not stop animating. "And then I will fix it. I work better with a deadline."

"That *is* a deadline. Production launch is a deadline."

"Production launch is not a deadline. Production launch is a *goal*. A deadline is when somebody is yelling at me. I have a file for those."

She showed me. It was a Trello board, on her personal account. It was titled *People Yelling*. It had three columns: *Yelling Now*, *Yelling Soon*, and *Yelling Eventually*. The Sales Manager appeared in *Yelling Soon* with a small frown emoji beside his name.

I had, by that point in my career, been on enough teams to know that this was either a deeply unhealthy workflow or a deeply honest one, and that it was probably both.

"Fine," I said. "But the API contract — when can we sit down and review it together? Because the shapes you've put in `pretend.json` are not the shapes the backend is going to return."

"They're close, though, right?"

"They are not close."

"They're vibes-close."

"They are not even vibes-close. The user object you've got has a field called `displayName`. The backend has `first_name` and `last_name`. You've got `lastSeen` as a string. The backend has it as a Unix timestamp. You've got `permissions` as an array. The backend has it as a comma-separated string in a column called `perms_csv`."

She looked, briefly, distressed. Then she rallied. She always rallied. Developer 1's spirit was, in this respect, a renewable resource.

"Okay," she said. "I'll write a transform layer. Adapter pattern. I can do it in a day."

"When."

"Soon."

"Developer 1."

"I'll do it tomorrow," she said, and I noticed, with a small internal flinch, that she said it the way I said it. The way the UX Designer said it. The way the UI Designer said it. The way, I was beginning to suspect, everyone on this team said it, including, most damningly, me.

I had read once, on a developer forum at three in the morning in the wreckage of a deploy gone bad, a comment that had stayed with me. The comment had said: *every "I'll do it tomorrow" is a small loan from your future self. The interest rate is ruinous and the collections agent is a Sunday night.* The comment had eight thousand upvotes. The replies had been a graveyard of teams who had taken those loans and could not pay them back.

I went back to my desk. I thought about saying something to the Tech Lead. I thought about pushing for an API contract review. I thought about, perhaps, just *writing* the contract myself, in a markdown file, in the next thirty minutes.

I did not. I told myself I had backend work to do. I told myself it would be presumptuous to dictate a contract to her. I told myself, in the end, the most dangerous thing of all:

She'll figure it out tomorrow.

She would not. None of us would. The transform layer would not get written. The shapes would not get reconciled. And on the morning of the beta, when the actual API would hit the actual frontend for the first time at any meaningful scale, the dashboard would render seventeen widgets, of which precisely one — the one that displayed the number 0 in a big font — would render correctly.

But that was tomorrow.

That afternoon, Developer 1 shipped a new animation in which the toast notification gently rotated three degrees on entry before settling. It was, genuinely, beautiful. The QA Tester filed a bug against it: *toast rotates on entry. is this intentional?* Developer 1 closed the bug as *Won't Fix — by design.*

I think, now, that this was the truest summary of our entire project. *Won't Fix. By design.*

— END OF 2.1 —

2.2 — BACKEND DEVELOPMENT

> "My PR sat in review for forty-one days. When the senior dev finally opened it, his first comment was 'this is a lot.' Yes. Yes, it was. It had been a lot for forty-one days."
>
> — anonymous backend engineer, blog post

I want to tell you that the backend, at least, was going well. I want to say that I, the careful one, the one with the notebook, the one who took the architecture diagram seriously, was holding the line.

I cannot.

The backend was, technically, being written. By me. On time. The streaming layer that the Tech Lead had insisted we build into a framework that did not support streaming was, against all odds, working. I had wrapped the legacy framework. I had duct-taped a Kafka consumer to the side of it. I had written, in the margin of my notebook, the words *this is a war crime against software engineering, but it works in dev,* and I had shipped it.

What I was not doing — and what would, in time, matter more than anything I did — was the unglamorous, social, project- shaped work that nobody had asked me to do and that I was, because nobody had asked, choosing to put off.

Every morning I would open my editor, and every morning I would see, pinned at the top of my notes, the same list:

TODO (me): - draft architecture doc - document streaming dependencies - propose CI pipeline shape - draft DB migration strategy - write API contract for frontend - chase tech lead for PR reviews

Every morning I would read the list. Every morning I would think *I should do at least one of these today.* And every morning I would, instead, pick up the next ticket in my own queue, because the next ticket in my own queue was a *coding* ticket, and coding was easy, and the things on the list were not coding, and the things on the list were *hard*.

This is the dirty secret of being a senior engineer. The hardest work is almost never code.

By the third week of the second month, my pull request queue looked like a small graveyard. I would scroll through it, sometimes, late at night, the way bereaved people walk through cemeteries.

#412 feat(auth): IdP token refresh opened 9 days ago #418 feat(stream): kafka consumer skeleton opened 6 days ago #421 chore(db): add indexes for portal opened 5 days ago #427 feat(export): job queue plumbing opened 3 days ago #431 fix(auth): edge case in claims map opened 2 days ago

Every one of them was *Awaiting review from the Tech Lead.*

I drafted, in our DM channel, a polite ping. I had the wording exactly right. Friendly without being passive-aggressive, specific without being demanding, light without being unserious. I had been workshopping it for, by then, about a week. I would type it. I would re-read it. I would imagine the Tech Lead reading it, mid-meeting, and frowning. I would imagine him replying *yeah I'll get to it tonight* and then not getting to it tonight, which would force me to send a second ping in three days, and a *third* ping in five, and at some point I would become, in his Slack pane, the Pinger Guy, the Person Who Pings, the small unwelcome notification icon he had begun to read as *hassling me again.*

So I did not send the message. I would close the draft. I would tell myself I would send it tomorrow.

This is what I mean. This is what I was doing. I was performing politeness as a costume for cowardice. I was, in my private language, *being respectful of his bandwidth.* In every other language — including the language his calendar was clearly speaking back to me — I was, in fact, allowing the project to quietly bleed out one unreviewed PR at a time.

The QA Tester walked past my desk one Wednesday. She looked at my screen, which had the PR queue open, which she had clearly seen me staring at for ten minutes.

"How long have those been sitting," she said.

"A while."

"How long is a while."

"Some of them are pushing two weeks."

She made a small noise. It was not a polite noise. It was the noise of a woman who had been right about everything for months and was no longer interested in being subtle about it.

"You have to ping him," she said.

"I have."

"In channel. In the open. So everyone sees it."

"That feels — confrontational."

"You know what's confrontational? Production going down because the auth refresh PR sat in review until everyone forgot why it was needed."

She walked off. She was, in some small terrible way, more of a project manager already than the Product Manager had ever been.

I stared at the queue. I thought about doing what she said. I opened the team channel. I started to type.

@tech lead — quick favor, could you take a look at #412 when you get a sec? It's blocking me on streaming and the IdP folks are starting to ask.

I read it. It was fine. It was professional. It was clear. It was the kind of message a competent person sends.

I deleted it.

I told myself I would send it after lunch. I went to lunch. I came back. I told myself I would send it after my one-on-one. I had my one-on-one. I came back. I told myself I would send it in the morning. I went home. I lay in bed.

I read once, on a developer forum, a thread titled *what's the longest a PR has sat unreviewed on your team*. The replies were like a wall of small, similar tragedies. *3 weeks.* *6 weeks.* *4 months and the original author had left.* *We have PRs from a previous reorg and we have collectively agreed not to look at them.* The top comment was a single line: *the PR queue is the truest org chart in any company. it shows you where decisions go to die.*

I lay in bed. I thought about my PR queue. I thought about the unsent messages. I thought about the architecture doc, still in *DRAFT*, with its single section titled *## 1. Overview* and its one finished sentence:

The Customer Portal is composed of:

That was it. That was the whole document. *Composed of:*. A colon, and then nothing. A promise made to no one and received by no one and never, in any meaningful sense, kept.

I reached for my phone. I opened the editor app. I pulled up the architecture doc. I added, after the colon, the word *services*. I saved the file. I felt, briefly, like I had done a thing.

I had not done a thing. I had typed a word. I knew the difference. I have always known the difference. Knowing the difference and acting on the difference are themselves two different things, and the gap between them, when measured in months, is the exact length of a software project that is about to fail.

I put the phone down. I closed my eyes. I told myself the line I would tell myself, in some form, every night for the next three months:

Tomorrow. I'll start it tomorrow.

The refrigerator hummed. The PR queue grew. Somewhere across the city, the Tech Lead was almost certainly playing a video game. Somewhere else, the Sales Manager was sending an email that began *just confirming, as discussed*, in which several things were being confirmed that had not, in any meaningful sense, been discussed.

And somewhere in a Kafka cluster I had stood up myself, my beautiful little wrapped streaming consumer was processing events flawlessly into a database table that nobody else on the team knew existed.

It was the best code I had ever written. It would also, in seven weeks, become the second worst thing that ever happened to me at work.

But that, of course, was tomorrow.

— END OF 2.2 —

2.3 — MOBILE APP DEVELOPMENT

"Our certificate expired during a five-day company offsite. The mobile app silently failed to receive notifications for the entire week. We found out from a one-star review that just said: 'silence.'"
— anonymous mobile lead, postmortem

Developer 3 was the mobile developer. There was, in some sense, only one of him, and in another sense — the sense that mattered — there were two of him, because he was building the same app twice, once for iOS and once for Android, and the two halves of him hated each other.

He sat at the desk closest to the window. He had two laptops, two phones, and a tablet. He wore noise-cancelling headphones that he never took off, which was, depending on who you asked, either a productivity hack or a cry for help.

In the kickoff, when the Tech Lead had drawn the boxes on the whiteboard, Developer 3 had pointed out — gently, almost apologetically — that the diagram did not include the mobile app.

"Right, yeah," the Tech Lead had said. "Mobile is — mobile talks to the same API."

"It does," Developer 3 had said. "But the way mobile *handles* the API — caching, offline, push, deep links — that's its own architecture."

"Sure," the Tech Lead had said. "Add a box."

Developer 3 had added a box. He had labeled it *Mobile*. He had drawn an arrow to *API*. He had stepped back and looked at the board and gone back to his seat. The whole thing had taken under thirty seconds. It had also been, I would realize later, the most attention the mobile app would receive from the team in the entire planning phase.

The thing about mobile is that it has its own problems. It has problems that feel, to people who don't work on it, like folklore. App store review processes. Provisioning profiles. Code signing certificates. Push notification tokens. Beta distribution channels. TestFlight slots. Internal tester limits. Two completely different sets of design guidelines, one of which insists on hamburger menus and the other of which insists, somehow, on the opposite. Things that take *minutes* on the web take *days* on mobile, because mobile is the only platform where Apple and Google can, at any moment, decide you are not worthy and reject your release for a reason that is, technically, real but, practically, deranged.

Developer 3 knew all of this. Developer 3 had been doing this for a long time. Developer 3 also knew, with the specific exhausted clarity of a person who had been the only mobile

developer on too many teams, that he could mention any of this in a meeting and watch the rest of the team's eyes glaze over in real time, like a window fogging up.

So he did not mention it. He stopped mentioning it after the second sprint. He nodded in standups. He said, every day, "Mobile is on track." And on his own time, in his own quiet way, he procrastinated on the one thing that would, in time, matter more than all the rest of his work combined.

He procrastinated on renewing the iOS distribution certificate.

This had been, originally, a calendar reminder. The reminder had said, in clear letters: *RENEW DIST CERT — EXPIRES MARCH 14.* He had snoozed it once, in February, because he was deep in the middle of refactoring the offline cache. He had snoozed it again, in late February, because the QA Tester had filed a bug about the splash screen flickering on Android tablets in landscape mode, and he had spent four days hunting that bug into a corner where he could shoot it. He had snoozed it a third time, in early March, because he had told himself *I'll do it Monday, certificates are a fifteen-minute job.*

Certificates are, in fact, a fifteen-minute job. They are also a fifteen-minute job that, if you do not do it, will, at one specific arbitrary moment, take down your entire mobile app for every single user, simultaneously, with no warning and no graceful failure mode. It is, in this way, the perfect crime against your future self. Cheap to commit. Catastrophic to discover.

I learned about it because I happened to be at his desk one afternoon, asking about something else.

"Hey, real quick," I said, "the API endpoint for the dashboard — do you guys want it paginated, or do you want everything in one payload?"

He turned slowly. He was wearing the headphones. He removed one ear cup. He looked at me with the specific bloodshot focus of a man who had been staring at Xcode for nine hours.

"Sorry," he said. "What."

I asked again.

"Paginate it," he said. "Mobile can't hold the whole payload in memory on lower-end Android devices. We'll cache pages."

"Got it."

He turned back. I noticed, on his screen, a notification that had clearly been sitting there for some time:

[Apple Developer] Your iOS Distribution Certificate will expire in 6 days. Renew now to avoid service disruption.

"Hey," I said, gently, "your cert."

"Yeah," he said. "I'll do it tomorrow."

"It expires in six days."

"I'll do it tomorrow."

"There's no rush."

"Exactly."

I did not push. I did not push *because* I did not push. There is no other reason. I told myself, *this is his domain, he knows mobile, he will handle it.* I told myself, *I have my own work.* I told myself, in the most pernicious of all the lies the team was telling itself that month, *it's fine.*

It was not fine.

Twelve days later, on a Tuesday, the certificate expired. The push notification service silently stopped working. The app did not crash. The app did not warn. The app simply stopped, gently, like an old radio losing signal. Notifications stopped arriving. Deep links stopped working. The login flow, which relied on a push for two-factor confirmation, broke for every new sign-in.

We did not notice. None of us were running the production build. The QA Tester was still on the dev build because the dev build was easier to test against. The Sales Manager demoed off a TestFlight build that did not use the same cert. For four days, the production mobile app did nothing, and nobody at the company knew.

We learned about it from the App Store. A one-star review appeared. It said, in its entirety: *the app is dead. nothing works. did your company also die.*

Developer 3 saw the review at lunch. He went very still. He read it twice. He did not say anything to anyone. He stood up, walked to his desk, opened his laptop, and renewed the certificate. The renewal took, as he had said it would, about fifteen minutes.

The fix took fifteen minutes. The damage took weeks. The trust — with the small but vocal mobile user base, with the App Store reviewers, with the team — took longer.

I read once, on a mobile developer forum, a thread titled *what have you put off that has bitten you the hardest*. The certificate stories were their own subgenre. They had a quality of myth. *I let the cert expire on Christmas Eve. I let the cert expire the day my daughter was born. I let the cert expire while I was at my own bachelor party. I have, now, three calendar reminders, two phone alarms, and a handwritten note in my wallet.* The replies went on for a thousand comments. They were funny in the way grief is sometimes funny.

That night, after the certificate was renewed and the notifications were flowing again, Developer 3 walked past my desk on his way out.

"Hey," he said. He stopped. He took a breath. "I should have done it when you said something."

"I should have said it twice," I said.

He nodded. We stood there for a moment, two professionals who had each, in our own quiet way, allowed something preventable to happen to a product we were both supposed to be protecting.

"Mobile is hard," he said, finally.

"Everything is hard," I said.

"Yeah," he said. He left.

I sat at my desk. I opened my notebook. I added a line, on a fresh page, all the way at the top:

the certificates always expire on a tuesday. the certificates always expire when you are looking the other way. the certificates do not care that you have had a long sprint. the certificates do not care about anything. that is what makes them certificates.

Then I wrote, underneath:

this is true of more than certificates.

I closed the notebook. I went home.

— END OF 2.3 —

2.4 — COMMITTING CODE

> "The commit message just said 'fix.' The diff was 4,800 lines. The author had left the company eight months ago. The line that broke production was inside that commit. We could not tell why it was there. We could not tell what it was for. We could only tell that, without it, six other things broke."
>
> — anonymous SRE, incident review

There is, on every team that has ever existed, a moment when the word *commit* stops meaning a small, careful thing and starts meaning whatever the developer happened to dump into git that afternoon. We had reached this moment by week eight. Possibly week six. The git history is, in a way, the only truly honest document a team produces, and ours had become the diary of a person who had stopped lying to themselves and was now, instead, simply not journaling at all.

The Tech Lead had set up the repo on day one. He had put up, on the team wiki, a page titled *Commit Message Conventions*. The page was a single bullet list. It said:

- Use Conventional Commits. - feat: for features. - fix: for bug fixes. - chore: for cleanup. - Be specific. Be concise. Future you will thank present you.

The page had been viewed, the wiki analytics showed, eleven times. Eight of those views had been the Tech Lead himself, checking that the page existed.

Here, for the historical record, is a representative sample of our actual commit messages from week eight:

feat: stuff fix wip more stuff ugh please fixing the thing that broke from the other thing revert the thing reverting the revert OK NOW nope actually now Developer 1: i hate this last commit was a lie sorry

There were three hundred and seventy-one commits between the start of sprint two and the end of sprint four. Forty-two of them had the message *fix*. Eleven had the message *wip*. Two had the message, mysteriously, only an emoji — once a fire, once a small skull. We did not, ever, find out which developer had committed the skull. We agreed, by silent consensus, not to investigate.

The DevOps Engineer tried, in the third sprint, to introduce a commit-message linter. He set up a pre-commit hook that would reject any commit that did not match the Conventional Commits format. It worked, technically, for about four hours. Then Developer 1 hit a critical merge conflict at 11:47 p.m., needed to commit the resolution to unblock CI, and discovered that her commit message of *please for the love of god* did not pass the linter. She did what any reasonable engineer would do. She bypassed the hook with `--no-verify`.

Within a week, *--no-verify* was the most popular flag in team history. The DevOps Engineer found it, in a long grep across the team's shell histories, used four hundred and twelve times. He stopped doing the grep after that. He told me, in the kitchen, that the result had made him feel "emotionally radioactive."

The pull requests were worse than the commits. The PRs were where the procrastination became visible to other people, and visibility, on this team, was a thing to be avoided.

A typical PR in week eight looked like this:

TITLE: Updates DESCRIPTION: (empty) FILES: 47 changed ADDITIONS: +2,140 DELETIONS: -890 REVIEWERS: (none) LABELS: (none) LINKED ISSUE: (none) AUTHOR: any of us, take your pick

The reviewer, opening this PR, would do what any sane engineer does in the face of a 2,000-line diff with no description: they would scroll through it for ninety seconds, find a typo in a comment, leave a single review comment that said *typo*, and approve the PR. This is the entire mechanism by which most software in the world gets shipped. I am not exaggerating. The proof is, well, all of it.

I was not innocent in this. My PRs were better than average, in the same way that being the tallest person in a sub-basement is taller than average. I wrote descriptions. I linked tickets. I tagged reviewers. But I did not, ever, push back when a 2,000-line *Updates* PR appeared in my queue with my name on it as a reviewer. I would open it. I would scroll. I would think *I cannot meaningfully review this.* I would find the typo. I would comment *typo*. I would approve.

Once, in a rare moment of conscience, I left a real review on one of Developer 1's PRs. It was a forty-line file with what I genuinely believed was a subtle race condition. I wrote three paragraphs explaining the problem. I suggested a refactor. I linked to a Stack Overflow answer that I thought was relevant. I hit *Submit Review*. I felt, briefly, like a real engineer.

She replied four minutes later: *good catch! will do this in a follow-up PR 🙏*.

There was no follow-up PR. There was never a follow-up PR. The race condition was, on the day of the beta launch, the cause of the second-largest production incident in the project's short and miserable life.

I read once, on a developer forum, a thread that asked: *what is the technical-debt commit you have written that you are most ashamed of?* The replies were extraordinary. They were a parade of small confessions, each one a little knife in the ribs of the next sprint.

> I committed a sleep(2000) into our payment retry loop because the test was flaky. It is still in production. It has been there for four years.

> I committed an `if (user.email === 'qa-test@example.com') return true;` so the QA team could log in. It is still in production. We have a real QA login system now. The if statement is also still there. It still works. We are afraid to remove it.

> I committed a try/catch around an entire module that just swallows every error. The comment above it says // TODO: handle properly. The git blame on that comment is from 2017.

The thread was a thousand comments long. The top comment, from a user with a name I have since forgotten, said:

every codebase is a wax museum of the moments its developers ran out of time. the older the codebase, the better the museum. the better the museum, the worse the maintenance bill.

I thought about that comment a lot. I thought about it the afternoon I committed, with my own hands, into the streaming consumer module, the line:

// TODO: this swallows every connection-reset error. // we should retry properly. for now, just log and move on. // — me, week 8, sorry future me

I committed it with the message *fix(stream): handle reconnect*. It was not a fix. It was a wound, dressed in a language that sounded like a fix. Future me would, in fact, be very sorry. Future me would, in fact, become Project Manager me, and Project Manager me would inherit the entire museum, with all its little wax figures of unkept promises, and Project Manager me would have to walk the halls of it with a clipboard and explain, to the team and to the clients and most of all to myself, why so many of the figures had my own face.

But that was tomorrow.

That afternoon, I closed my laptop. I went home. I lay in bed. I told myself I would go back, in the morning, and replace that TODO with a real fix.

I did not.

I forgot about it for seven weeks.

When I remembered, the team was on fire and the connection- reset error was the thing burning hottest, and the comment *— me, week 8, sorry future me* was waiting for me in the production logs at three in the morning, exactly where I had left it, exactly as honest as I had been when I wrote it, which was to say, exactly as cowardly.

— END OF CHAPTER 2 —

Chapter 3 — Quality Assurance

3.1 — UNIT TESTS

> "Our test coverage was 11%. Of that 11%, half were tests of our own test helpers. We were testing the testing. We had achieved recursion. We had not achieved testing."
>
> — anonymous QA engineer, retro slide

The QA Tester sent a message in the team channel one Monday in the third month. It was the kind of message you can tell was written calmly only after several drafts of much less calm messages had been deleted.

QA TESTER: hey team. quick check-in on test coverage. i pulled the report this morning. we are at 11%. target was 70% by end of sprint two. we are now in sprint nine. would love to find a time this week to align on a path forward 🙏

The 🙏 was, I would learn later, a lie. There was no plea in her heart. There was, instead, a slow-burning fury that had been gathering since approximately sprint two, when she had first asked, in her gentlest possible voice, whether people were writing tests, and had received, in response, seven slightly different shapes of *yes*.

The shapes of *yes*, in chronological order, had been:

Developer 1: yeah totally i'll add some this week Developer 3: mobile is hard to unit-test, i'll write integration ones instead Tech Lead: we're doing TDD, so technically every feature has tests Me: yes (this was a lie) DevOps: i don't write features, i write infra, so the question doesn't apply UI Designer: i don't write code? sorry? Sales Mgr: what

She had taken these answers and she had filed them, in her notebook, under a heading that said *RED FLAGS, SOFT*. She had then gone and pulled the actual coverage report. She had filed *that* under *RED FLAGS, ARTERIAL*.

She had, to her credit, not panicked. She had, instead, spent the next six sprints quietly trying to get the team to write tests, in increasingly direct language, and the team had, with escalating creativity, found ways to not write tests.

The reasons for not writing tests, on our team, fell into a small but consistent set of categories. I had been keeping a list, partly out of professional curiosity and partly out of the dawning horror that I was hearing my own voice in some of them.

- "I'll add tests in a follow-up PR." (We have not yet, in the history of this team, shipped a follow-up PR consisting of "the tests I owed you from last sprint.")

- "This code is too simple to test." (Said about a 400-line function with seven nested conditionals and a cache.)

- "This code is too complex to test." (Said about literally the next function in the same file.)

- "Mocking this would take longer than writing the actual code." (True, sometimes. Used to justify not testing things that could trivially be tested.)

- "We have integration tests." (We did not, in any meaningful sense, have integration tests.)

- "QA will catch it." (The thing that was caught, in this case, was the QA Tester's last nerve.)

- "I'll do it tomorrow." (Reader, you know.)

The QA Tester scheduled a meeting. She titled it, with a restraint I admired, *Test Coverage Sync.* She invited the whole team. She prepared a single slide. The slide had three words on it, in 96-point font, centered:

ELEVEN PERCENT

When we filed in, the slide was already up. She did not say anything for a long time. She let the slide do the talking. The slide did, in my opinion, an excellent job.

The Tech Lead broke first. "I mean — eleven percent of *what*, though. Like, statement coverage? Branch coverage? It depends on how you count."

"All of those," she said. "All ways of counting. It is bad in every dialect."

"There's also coverage we get from manual testing —"

"Manual testing is what *I* do. It is not coverage. It is me clicking on buttons until my carpal tunnel acts up. It does not run in CI. It does not catch regressions. It does not exist in any form a future engineer can read."

There was a silence. The slide remained, accusing, in 96-point.

"Look," Developer 1 said. "We've been moving fast. We knew we'd come back and add tests later. That's fine. That's normal."

"It is not normal," the QA Tester said, quietly. "It is *common*. Those are not the same."

I felt something small and sharp inside my chest. *It is common, not normal.* I have, in my life, had perhaps three sentences fall on me with the weight of a verdict, and that was one of them. It applied to so many things on this team. The PR queue. The architecture doc. The cert. The TODO in the streaming consumer. *It is common, not normal.*

The Tech Lead tried to rally. "Okay. So let's commit to a coverage push. Sprint ten. Everyone writes tests for the features they shipped. We'll get to fifty percent."

"By when."

"End of sprint ten."

"Are you going to enforce it."

"In the sense of —"

"Are you going to *block PRs* without tests. Are you going to *fail builds* without tests. Are you going to make *any* mechanism, beyond your own asking-nicely, that creates a consequence for shipping untested code."

The Tech Lead blinked. He had not expected this question. He was going to give one of his usual answers. *We'll keep an eye on it. We'll talk in standup. I'll send a reminder.* He opened his mouth.

She stared at him.

He closed his mouth.

"I'll set up a coverage gate in CI," he said, finally. "It'll fail builds below sixty percent."

"By when."

"This week."

"Promise me."

"I promise."

He set it up that Friday. He set the threshold to sixty percent. The first build after the gate was enabled failed because, of course, the codebase was at eleven percent. The gate was, by the following Monday, set to *warn but not fail*. By Wednesday, it had been set to *report but not warn*. By Friday of the next week, the entire job had been *temporarily disabled* while the team "worked on getting coverage up."

The job was never re-enabled. The coverage never went up. *Temporary*, in our team's vocabulary, had become a tense of its own — a verb form meaning *forever, but with deniability*.

I read once, on a developer forum, a thread titled *the coverage gate that didn't*. The replies were a flood of small, identical stories. Teams that had set thresholds. Teams that had lowered them. Teams that had disabled them. Teams that had moved them into a second pipeline. Teams that had moved the second pipeline into a manual-only run. Teams that had moved the manual-only run into a wiki page that said *we should run this sometime*. The wiki page, in every story, had no recent edits.

The top reply was: *test coverage is the only number in our industry that, when it goes up, means something is wrong. Either you're writing useless tests to game the gate, or you're rewriting your code in a way you wouldn't have if the gate weren't there. Coverage is not a measurement. Coverage is a behavior change. Treat it that way or don't bother.*

I thought about that line a lot. I think about it still. *A behavior change.* Of course. We had not failed to write tests because writing tests was technically hard. We had failed to write tests because we had not, as a team, agreed to be the kind of team that wrote tests. The eleven percent was not a coverage number. It was a portrait.

The QA Tester walked back to her desk after the meeting. I caught up with her in the corridor.

"Eleven percent," I said. "Brutal."

"Yeah."

"What do you want me to do."

She stopped. She turned. She looked at me with a fatigue that went all the way down.

"I want you," she said, "to write the tests for the streaming consumer you wrote. You know, the one I have been asking you about, in our one-on-ones, every two weeks, for fourteen weeks."

I had, in fact, been asked. I had, in fact, every time, said *yes, I'll do it next sprint.*

"Right," I said. "Yes. I'll do it tomorrow."

She did not say anything. She looked at me for a long second. She nodded, once, and walked away.

I have been thinking, ever since, about what that nod meant. I think it meant: *I have heard that sentence from this team two hundred and forty-one times and I no longer believe any of you.*

I do not blame her.

I would not believe me either.

— END OF 3.1 —

3.2 — SOFTWARE TESTS

> "There were two hundred manual test cases for the feature we were about to launch. The morning of launch, the PM cut the feature. The cases sat in the test management tool, untouched, for years afterward, like graves nobody had told."
>
> — anonymous QA lead, hallway conversation

If unit tests were the part of testing the developers ignored, software tests — by which I mean the broader universe of integration tests, manual tests, exploratory tests, regression tests, end-to-end tests, and the strange hybrid creature called *user story validation* — were the part of testing the developers had never even heard of.

This work fell, in its entirety, on the QA Tester. There was one of her. There were seven of us. The math was, on paper, absurd. In practice, it was worse.

The way it worked, in our team, was this. A developer would finish a feature. They would, in their PR description (when they wrote one), put a single line that said *please test*. The QA Tester would receive a notification. She would open the PR. She would scroll through the diff to try to understand what had changed. The diff would be, often, opaque to her — not because she couldn't read code, but because the developers had, in their commits called *fix* and *stuff* and *ugh*, refactored seventeen things at once, and there was no description of which thing was the actual feature.

She would then build the feature locally. The local build would fail, because nobody had updated the local setup documentation since week two. She would ping the developer. The developer would say, *oh yeah, you need to set this env variable, also delete this folder, also run this migration*. She would do these things. The local build would still fail. The developer would say, *huh, works on my machine.*

This phrase — *works on my machine* — was, on our team, said with such frequency that the DevOps Engineer had, half- seriously, ordered eight stickers that said WORKS ON MY MACHINE in the style of an inspection certificate, with a place at the bottom for the developer to sign and date. He kept them in his desk drawer. He handed them out, deadpan, when the moment called for it. By month four, all eight had been distributed. By month five, three of them were stuck on Developer 1's laptop.

The QA Tester would, eventually, get the build running. She would test the feature. She would file bugs. The bugs would be, in roughly equal proportions, cosmetic, functional, and existential. (The existential ones were her specialty. They were bugs that asked questions like *what is supposed to happen when a user with no permissions clicks Export*, or *should the dashboard render if the API returns 401*, or, my favorite, *when is a session over*.)

She would assign the bugs back to the developers. The developers would mark roughly half of them as *Won't Fix — out of scope* and roughly half as *Will Fix in follow-up*. The follow-ups would not fix them. The bugs would migrate to a backlog called *To Triage*. *To Triage* would grow, sprint over sprint, like a slow tide. By month four, *To Triage* contained four hundred and seven items. The QA Tester had stopped opening it. She referred to it, in her notebook, as *The Drowned*.

I was responsible for one of the worst incidents in this part of our story. I owe it to the record to confess.

In sprint seven, I refactored the authentication middleware. I did so because I had, finally, in a fit of late-night clarity, realized that the version on the main branch did not correctly handle token refresh, and that, when a client's session was about to expire, the user would experience a silent logout, and that this was unacceptable.

The refactor was clean. It was, in my biased opinion, elegant. It was forty-three lines of TypeScript that replaced one hundred and eighty-eight lines of TypeScript. It worked beautifully on my machine. It passed all of my (almost nonexistent) tests. I committed it. I opened a PR. I tagged the Tech Lead. I waited.

The Tech Lead did not review the PR. (You will, by now, have guessed this.)

The PR sat. I waited four days. I sent a polite ping. The ping was ignored. I waited another four days. I sent a less polite ping. The Tech Lead replied, *yes will look tonight sorry*. He did not look tonight. He did not look the next night.

I had, in my project, a deadline I had committed to. The streaming work depended on the auth refactor. I needed to unblock myself. So, in week ten, I did the thing that you should never, ever, ever do. I got tired of waiting. And I merged my own PR.

I rationalized this in the way you rationalize anything at ten o'clock at night when you have a deadline and you are the only person at your desk. I told myself: *the Tech Lead isn't reviewing it anyway. it's a clean refactor. it'll be fine.*

I did not tell anyone I had merged it. I did not announce it in the team channel. I did not even update the PR description to say *self-merged after waiting two weeks for review.* I just merged it. I went home. I slept the sleep of the self-justified.

Three days later, the QA Tester filed a bug. The bug was titled, with no preamble: *every login is broken*.

It was not, technically, every login. It was every login that used a particular OAuth flow, which was approximately seventy percent of our users, including all three pilot clients. The auth refactor had subtly broken the claims-mapping for that flow. My tests had not caught it because I had not written tests for that flow. The Tech Lead's review would have caught it because he was the one who had originally written that flow and would have flagged it on sight.

I had skipped both safety nets. With my own hands. With a small, clear thought of *it'll be fine.*

The hour after that bug landed in our channel was the worst hour of my professional life up to that point. I sat at my desk. The QA Tester walked over. She did not say anything. She just stood there. The Tech Lead walked over. He looked at my screen. He looked at the bug. He looked at me.

"When did you merge that PR," he said.

"Tuesday."

"Tuesday."

"You hadn't reviewed it. I needed it for the streaming work."

He took a long breath. He was not angry. He was, somehow, worse than angry. He was *disappointed*, in the specific quiet way of a person who has just realized they were also part of the problem.

"I should have reviewed it," he said.

"I should have waited."

The QA Tester said, with a calm I will remember for the rest of my life, "You should both have done a lot of things. Right now you're going to help me triage this. I have eleven other bugs from this morning and three of them might be related."

We worked through the night. We rolled back the auth refactor. We patched the immediate bug with a duct-tape solution that, naturally, I committed with the message *fix(auth): claims map regression*. We re-released.

I went home at five in the morning. I lay in bed. I did not sleep. I thought about a thread I had read once, on a developer forum, titled *what's the worst thing you ever did to your QA team without realizing it*. The replies had been a small, sad litany. *I refactored the API right before freeze.* *I deleted a feature flag without telling them.* *I "just renamed a few endpoints" the day before regression.* *I merged my own PR because no one would review it.*

The top reply, the one I had bookmarked at the time and forgotten about, was a single sentence:

QA does not test your code. QA tests your communication. when you skip QA, you are not skipping testing. you are telling the team you don't need them. they hear it.

I lay in bed. I thought about that. I thought about the QA Tester standing at my desk, saying nothing. I thought about the way she had not, even once, raised her voice. I thought about how she had been right about every single thing she had flagged, for months, and how every single one of us had, through our own polite-and-overworked excuses, made it more expensive for her to do her job than it ever should have been.

I did not sleep that night. I am not sure, on a fundamental level, that I have, in any meaningful sense, slept since.

— END OF 3.2 —

3.3 — TEST AUTOMATION

> "The end-to-end test suite was a single Selenium script that clicked through the login flow. It had been written in 2018. It ran every night. It had passed every night. We discovered, in 2023, that it had been pointed at a staging environment that had been decommissioned in 2020. It had been passing because it was failing too quickly to fail."
>
> — anonymous engineer, postmortem doc

The DevOps Engineer was, by trade and by temperament, a person who believed in automation the way some people believe in God. He believed that everything that could be automated should be automated, and that the things that could not be automated should be re-examined to see whether, perhaps, with the right amount of effort, they could in fact be automated. He had, in his first week, automated the act of remembering team birthdays. He had, in his second week, automated the act of automating birthdays for new hires. He had, in his third week, automated the act of generating the report that showed how many things he had automated.

He was, as you can perhaps tell, the wrong person to put in charge of automating only some things. He needed, for his own peace of mind, to be the person who automated all of them. We did not give him this. We gave him, instead, the specific kind of half-trust that lets a competent engineer build half of a thing and then be slowly killed by the half of it that doesn't exist.

Test automation, on our project, was a perfect case study in this.

The plan, in week one, had been the following. Sprint two: DevOps would set up a CI pipeline. Sprint three: he would plug in a unit test runner. Sprint four: he and the QA Tester would stand up an end-to-end test framework, run from CI, against staging. Sprint five: the team would write end-to-end tests for the critical user journeys. By sprint six, every PR would run through the suite before merging.

The plan, in actuality, went like this. Sprint two: DevOps set up the CI pipeline. (He delivered this on time. It was the only thing on the project that was ever delivered on time. Nobody noticed.) Sprint three: he plugged in the unit test runner. It immediately started failing because there were almost no unit tests, and the ones that did exist had been written against an old version of the codebase and no longer compiled. He filed a ticket. The ticket was assigned to *the team*. The team did not fix it. The job was, by sprint four, marked as *known to be flaky, ignore for now*.

Sprint four came and went without an end-to-end framework. DevOps had, in his own words, "started looking into it." He had, specifically, opened seventeen browser tabs about

Playwright vs. Cypress vs. WebdriverIO, read the first paragraphs of all of them, and then closed all the tabs at once during a Friday afternoon focus crash.

He had then, the following Monday, sent a message in the team channel:

DEVOPS: hey team — for E2E, are we going Playwright or Cypress? leaning Playwright but want to align.

There were no replies.

He sent it again, three days later, with a 👀 emoji.

There were no replies.

He sent it a third time, a week later, with the addition of:

DEVOPS: ok i'll just pick. going Playwright. setting up a skeleton this week.

This time there was one reply, from Developer 1: *cool* and a 👍.

He set up the skeleton. It was, he later told me, *beautiful.* It had a config. It had a fixtures folder. It had a single example test that opened the login page and verified the title was correct. He committed it. He opened a PR. He tagged the entire team. He wrote a description that began with the words *please add tests as you ship features.*

The PR sat for nine days. Nobody looked at it. Eventually he self-merged it. He felt, he said later, *the silent shame of being correct about something nobody cared about.*

The skeleton sat in the repo, unloved, for two months. The example test continued to pass, every night, alone, like a lighthouse on an unvisited coast. No one added a second test. The QA Tester, when asked about it, would say only: *I have been writing manual test cases by hand for fourteen weeks. I do not have the time to also build the automation that should have been built four months ago. I am one person.*

There is a thing that happens to test automation work on teams like ours. It becomes everyone's responsibility, which means it becomes nobody's responsibility, which means it becomes the responsibility, eventually, of the one person foolish enough to volunteer, who then becomes resentful, and then quiet, and then, in the worst cases, no longer employed at the company.

I had read once, on a developer forum, a thread that asked: *who, on your team, wrote the test automation suite?* The replies fell into two clean halves. Half were from the authors of the suites — engineers who described, in tones ranging from melancholy to outright bitter, how they had spent six months building the suite and how nobody had written a test in it after them. The other half were from team members who had inherited a test suite — engineers who said, with the perfect honesty of people two jobs removed from the original sin, *I don't know who wrote it. They left before I joined. The suite is named after a Greek god. None of us know why.*

The top comment on the thread had been: *test automation is not a technical investment. it is a cultural one. you cannot automate your way into a team that does not believe in testing. you can only build the road and watch them not walk on it.*

By month four, the road existed, and we were not walking on it.

The Big Merge — the one I will tell you about properly in its own scene, the one where everything we had been deferring came due in the same morning — was scheduled for the start of month five. The entire feature was supposed to be merged into a release candidate, against which we would then run a full battery of tests, identify the gaps, fix them, and ship the beta.

You can guess what happened. The Big Merge failed in CI in seven different ways. Five of those ways were unit-test failures from tests that didn't exist. The build pipeline ran for forty-six minutes before failing on a missing secret. The end-to-end test suite, which was supposed to have grown to dozens of tests by then, contained one test — the original lonely lighthouse, which had been written by DevOps himself — and it passed. It passed against a staging environment that, it turned out, was running a version of the app that was nine sprints old.

For three days, every CI run was red. The DevOps Engineer worked through the nights. He patched. He bypassed. He duct-taped. He set up new pipelines. He skipped tests. He did, in three days, what should have been three months of team work, and he did it alone, in a hoodie, on a couch in the corner of the office, eating cold takeout in a way that suggested he had stopped tasting food some time ago.

I went over to his couch on the third night. It was about two in the morning. He looked up. His eyes were two small red puddles.

"Hey," I said.

"Hey."

"How can I help."

He laughed. It was a strange laugh — not bitter, not even sad. Just *worn*. Like a sound a thing makes when it has been bent past its tolerance.

"You can help," he said, "by writing the tests we should have written four months ago. But you can't help with *tonight*. Tonight is just me. Tonight is what tonight is because of all the other tonights that should have been something."

I did not have an answer for that.

He went back to his laptop. I sat with him for another hour, doing nothing, just being there. We did not talk. The build ran. The build failed. The build ran again. The build failed again. At some point, just before dawn, the build passed. He stared at it. He did not celebrate.

"Don't trust it," he said.

"What."

"Don't trust the green. Half of it is skipped. The other half is testing the wrong things. We have a green build now, but it doesn't mean the software works. It means the pipeline doesn't fail. Those are not the same."

I went home as the sun came up. I thought about that sentence — *we have a green build, but it doesn't mean the software works* — for the entire walk. I thought about all the green checkmarks on all the dashboards I had ever seen, in my whole career, and I wondered, for the first time, how many of them had been like ours: not signs of a healthy system, but the slow-blinking lights of a smoke alarm whose battery had been removed years ago.

The beta would ship that Friday. It would ship with a green build and a broken product. The two had, on this team, become entirely different things, and we had not known the distinction until it was too late to use it.

— END OF 3.3 —

3.4 — SECURITY TESTS

> "The penetration tester logged in as 'admin/admin' on day one and pulled the entire customer database in twelve minutes. He asked us, very kindly, whether we wanted him to keep going. We said no. He said, very kindly, that he was going to anyway. That was when we knew the launch was going to be moved."
>
> — anonymous engineering manager, retro

The Security Engineer arrived on a Tuesday. He arrived four days before the beta launch. He arrived because the Sales Manager had, in a moment of unexpected diligence, mentioned in a client call that we did "third-party security review" of all releases, and one of the clients had, with a polite smile, asked to see the report.

There was, of course, no report. There had not been a third- party security review. There had not been a *first*-party security review. The closest thing we had to a security review was that, in sprint two, the Tech Lead had said, in passing, "we should think about security at some point," and nobody had disagreed, and we had all then proceeded to not think about security.

So the Sales Manager, in a panic, had hired a contractor. The contractor was the Security Engineer. He had been available on short notice, which is the kind of detail nobody on the engineering team had thought to interrogate at the time, and which, in retrospect, should perhaps have worried us. Security engineers who are available on four days' notice are either between gigs or actively waiting for the precise opportunity to be confronted with a team like ours. The Security Engineer, it would turn out, was the second kind.

He arrived in a black hoodie. He did not, despite this, look like the stereotype of a hacker. He looked like a tax auditor who had recently been told that he was, in fact, allowed to enjoy his job, and was still adjusting to the news. He had a small notebook with a black cover. He carried a laptop with at least eleven stickers on it, several of which I did not recognize and one of which, I was fairly sure, was a logo for an organization that, depending on who you asked, either did not exist or was very deeply not supposed to.

He sat down at the small desk we had cleared for him. He plugged in his laptop. He looked around. He took a sip of coffee. He opened a terminal. He cracked his knuckles. He said, conversationally, "Walk me through your auth flow."

The Tech Lead walked him through the auth flow. The walk- through took, as nearly as I can reconstruct it, six minutes. By the end of those six minutes, the Security Engineer had filled approximately one and a half pages of his small black notebook. He had not, in any visible way, reacted to anything. He nodded throughout. He occasionally said, "Mm." He once said, "Interesting." He said it the way a doctor says it when they have just looked at a chart and do not, professionally, want to alarm the patient.

When the Tech Lead finished, the Security Engineer asked, "Can I have a test account?"

We gave him one.

He logged in. He clicked around for about thirty seconds. He stopped. He looked at his screen. He looked at us. He said, with a kind of reverent quietness, "You have no rate limit on the login endpoint."

"Right," the Tech Lead said. "We were going to add one in the polish phase."

The Security Engineer said nothing. He typed for ninety seconds. He turned his laptop. On the screen was a small script. The script had, in the previous ninety seconds, attempted approximately eighteen thousand logins against our staging environment using a list of common passwords. Of those eighteen thousand attempts, four had succeeded. The four accounts that had been compromised were, the Security Engineer noted, accounts the QA team had created for testing, with passwords like *Test1234* and *Password!* He had pulled their full session tokens. He showed us the tokens. He showed us, using the tokens, the dashboard those users would have seen. The dashboard had real-looking data on it because it was, in fact, real data, because nobody had thought to populate the test accounts with fake data.

"Okay," the Tech Lead said.

"There's more," the Security Engineer said, with a small patient smile.

There was more.

For the next two hours, the Security Engineer walked us through, with the methodical politeness of a man giving a museum tour, the security posture of our application. He showed us:

- That our export endpoint, designed to let users download their own data, had no authorization check on the user- ID parameter. He had downloaded another user's full transaction history by changing a number in a URL.

- That our dashboard widgets, when given an unexpected query parameter, returned full SQL error messages, including the names of every table in the database, the schema, and, in one case, a sample row.

- That our session cookies, which we had set to *not* expire because the Product Manager had once said "session expiry is annoying for users," remained valid for as long as a user had been signed in, even after they had clicked Log Out, because the logout endpoint only cleared the cookie on the client and did not invalidate the session on the server.

- That we had, in a config file in the repository, the test API key for our payment processor. The key was "test" but it had access to a sandbox that, the Security Engineer noted with the expression of a man delivering very bad news very gently, contained real-looking customer information from another team.

- That our admin panel, which we had deployed at `/admin`, was protected by a single check that read, in source: *if (req.query.admin === 'true')*. He had, naturally, found this in the bundled JavaScript. He had, naturally, accessed the admin panel. He had, naturally, taken a screenshot.

- That the JWT we issued to authenticated users was signed using an algorithm called *none*, which is, as the Security Engineer explained, with the careful enunciation of a man who could not believe he was having to explain this in 2026, an algorithm in which the signature is *not actually verified*. He had forged a token. He was, technically, the CEO. He showed us the dashboard the CEO would have seen. We had not, until that moment, realized that there was a CEO dashboard. There was. It contained financials. It was, the Security Engineer noted, "a real document."

I want to write the rest of this scene as a comedy. I want to find, in the wreckage, the kind of dry workplace humor that would let me put a small joke at the end of a sentence and lighten the load.

I cannot. There is no joke in the *none* algorithm. There is no joke in another team's customer data sitting in a sandbox we had access to. There is no joke in the moment, two hours into the review, when the Tech Lead, very quietly, asked, "Do we — do we report these? Like, externally?" and the Security Engineer said, "If any of this had been in production with real users, you would have been reporting the breach to a regulator within seventy-two hours."

The launch was four days away.

The room was very quiet. The Security Engineer closed his notebook. He looked at us. He had the face of a man who had done this enough times that the shape of the conversation he was about to have was not new to him.

"Look," he said. "This is fixable. You have three days. We prioritize. Nobody panics. We pick the top five, we ship patches by Friday, we delay the items we cannot fix. The beta moves by a week. Everyone takes a breath. This is, in the scheme of things, normal."

"Normal," the Tech Lead repeated.

"Common," the Security Engineer said. He paused. "Not normal."

He used, without knowing, the same distinction the QA Tester had used in 3.1. I think now that this is the distinction I have come back to most often, in my career, since this project. *Common, not normal.* The version of this distinction that the Security Engineer was offering had a specific meaning. It meant: *yes, lots of teams ship software that is this insecure. No, that does not make it acceptable. The fact that you can find your situation described in a thousand postmortems does not mean your situation is acceptable. It means the industry has, for a long time, agreed not to look directly at how bad this is.*

I read once, on a security forum, a thread titled *the moment you knew you needed to delay launch*. The replies were extraordinary. Many were variants of our story — contractors brought in too late, hardcoded keys, missing auth checks, JWTs signed with *none*. The top reply, the one I bookmarked the night of that meeting and have re-read more than once, was a single line:

the security review you skipped in week two is the emergency you will run in week twenty-two. it will cost you the launch date. it will not, however, cost you anything close to what *not* running it will cost you. pay early. always pay early.

The Security Engineer left at six p.m. He shook each of our hands with the same calm courtesy. He said, to the Tech Lead, "I'll send the report tonight. Read it. Send me the list of items you'll fix by Friday. We'll do another pass Sunday. Eat something. Sleep if you can."

After he left, the team sat at the table for a long time. Nobody said anything. The Sales Manager arrived at the door. He read the room. He turned around. He left without speaking.

The launch would be moved by ten days. The Sales Manager would have the worst three client calls of his career explaining why. The Tech Lead and I would not sleep in any meaningful sense for the rest of the week. The DevOps Engineer would patch the JWT library and add server-side session invalidation in a single, terrified, nineteen-hour sprint. Developer 1 would weep, briefly, at her desk, because she had been the one who had committed the *if (req.query.admin === 'true')* check, and she had genuinely believed it was a placeholder she would clean up later, and "later" had, as it always does, become *too late*.

I would, on the Friday, write the unit tests for the streaming consumer that I had been promising the QA Tester for fourteen weeks. I would write them in a single five- hour sitting, quietly, at my desk, and I would commit them without telling anyone, and the QA Tester would notice them in the next morning's CI report, and she would walk past my desk, and she would not say anything, but she would put her hand on my shoulder, briefly, as she went by, and that would be the kindest thing anyone did for me that month.

We were not yet at the Mirror Moment. The Mirror Moment was two scenes away. But I knew, that Friday night, sitting in the office with the Tech Lead and the DevOps Engineer and the cold pizza, that something inside the project had broken, and that the breaking had not been caused by the Security Engineer. The Security Engineer had merely turned on the lights.

The breaking had been caused by us. By me. By every "I'll do it tomorrow" any of us had ever said about anything that did not feel, at the time, like a feature.

Security, I understand now, is what you do when nobody is asking you to do it. By the time someone is asking, you are already late.

— END OF 3.4 —

3.5 — PERFORMANCE TESTS

> "The page took eleven seconds to load on a real user's machine. On the developer's machine, behind a corporate gigabit connection, on a $4,000 laptop, with the dataset of one fake user, it took 200 milliseconds. The developer was furious that anyone would call the page slow."
>
> — anonymous tech lead, blog comment

The performance test happened on a Wednesday in month five, and it happened, as so many catastrophes do, by accident.

The DevOps Engineer had been setting up a load test environment, more out of personal curiosity than any official mandate. He had stood up, in a corner of our staging cluster, a copy of the database that had been populated with realistic data volumes — about a million users, thirty million transactions, a year of dashboard events. He had pointed a small load-testing tool at the staging API and had asked it, casually, to simulate two hundred concurrent users hitting the dashboard.

He had walked away to get coffee. He had been gone for about eight minutes. When he returned, the load tester had crashed, the staging API had crashed, the staging database had a CPU graph that looked, he later said, "like a heart attack drawn by a child," and three Slack notifications had piled up from the cloud provider's billing alert system.

He had, with the calm of a man who had seen this kind of graph before, taken a screenshot of the dashboard latency panel and posted it in our team channel. The screenshot showed, for a single dashboard request:

p50: 8.2 seconds p95: 23.4 seconds p99: 47.1 seconds (timed out) errors: 18%

Underneath the screenshot, he wrote:

DEVOPS: ran a casual load test against staging with prod-shaped data. dashboard is, uh, slow. anyone want to talk about that six-table join.

I read the message. I felt, somewhere in my chest, the small specific cold feeling of being recognized in a crime scene.

The six-table join was the query I had warned the Tech Lead about in week six. I had run it against a snapshot of realistic data. I had told him: *eight seconds per widget, nine widgets, this is going to be brutal.* He had said: *we'll add a materialized view, or a cache, later. polish phase.*

I had accepted that answer. I had accepted it because it was easier than pushing back. I had not written a follow- up ticket. I had not raised it again. I had let it slide into the same drawer where all the other things had gone: the architecture doc, the dependency list, the API

contract, the unit tests, the security review, the cert renewal, the design tokens. The drawer had a name, and the name was *later*, and the drawer, like all of them, had no bottom.

Now, in week twenty, with the launch already pushed, with the Security Engineer's report still circulating like a small grim weather system, the drawer had opened, and out of it had come a database query that took forty-seven seconds to time out under a load that any real customer would, on day one, exceed.

I went to the Tech Lead's desk. He was already looking at the message.

"Eight seconds," he said.

"Per widget."

"Per widget. Right."

"There are nine widgets."

"There are nine widgets."

"On the dashboard that the CFO of client #2 will load on Monday morning."

"On the dashboard that the CFO of client #2 will load on Monday morning."

We were repeating each other, like two cartoon characters slowly realizing the situation. There is something about performance problems that does this to people. They are unlike functional bugs. A functional bug is a thing you can find and fix. A performance bug is a *property* of the system. It is in the architecture. It is in the data shape. It is in the way the joins are joined and the indexes are indexed and the queries are written. A performance bug at twenty seconds per request is not a bug. It is a *design choice*, and the design choice is already in production by the time you notice.

The QA Tester had walked over by then. She read the screenshot. She did not say anything. She did not need to. She had, in sprint three, opened a ticket called *PERF: load test the dashboard query with prod-shaped data*. The ticket had been put in the backlog. The backlog had, through the magic of agile prioritization, never quite gotten around to it. The ticket was, that morning, three months and eleven days old.

"I told you," she said, mildly. She did not say it cruelly. She said it almost gently, the way a parent says it to a child who has, against all advice, touched the stove.

"You told us," the Tech Lead said.

"I told you."

"You told us in writing. In a ticket. With a number on it. Three months ago."

"Yes."

"I'm sorry."

She nodded. She walked away. She did not need to say anything else. The ticket had said everything.

The DevOps Engineer pulled up the query plan for the six-table join. We gathered around his screen. The query plan was, as he put it, *expressive*. It contained the phrase *Seq Scan* on three tables that should have been indexed. It contained the phrase *Hash Join* on a table that had, the DevOps Engineer noted, more rows than there were people in our state. It contained, mysteriously, a *Sort* operation on a column that was not in the SELECT list. None of us could explain the Sort. We agreed, by silent consensus, not to look at it too closely.

I had read once, on a developer forum, a thread titled *the slow query that took down your launch*. The replies had been a parade of small, perfect tragedies. *Our search endpoint did a SELECT * on a table with seventy million rows.* *Our login flow ran an N+1 query to populate the user's permissions, one query per permission.* *Our dashboard called the same endpoint seventeen times because each widget thought it was the only one rendering.* *Our analytics query took thirty seconds and was called on every page load. Yes, every page load. No, we did not know.*

The top reply on that thread was, as it so often is on that forum, a single sentence that has stayed with me:

every slow query in production was a fast query in development. the bug is not in the query. the bug is in the gap between the data you tested with and the data your users actually have.

I understood, that Wednesday afternoon, looking at the query plan, the depth of that sentence. We had not failed to test for performance because performance was hard. We had failed to test for performance because we had been testing against `pretend.json`, against three users and twelve transactions, against the cheerful little staging universe in which everything is fast because nothing is real.

The fix was not a single fix. The fix was an architecture change. We needed:

- a materialized view of the reporting table, refreshed every five minutes - indexes on three columns that should always have been indexed - pagination on the dashboard widgets, which had been designed to load *all* of a user's data at once because, in development, all of a user's data was twelve rows - a cache layer in front of the API so that nine widgets requesting the same underlying data didn't each cause nine database queries - a frontend change to debounce the auto-refresh, which Developer 1 had set, charmingly, to two seconds, because she liked the way the loading skeletons breathed

The work would take, the Tech Lead estimated, three weeks. The launch was, at this point, ten days away.

The Tech Lead and I sat at his desk for a long time. We did not say anything for a while. We were, both of us, doing the exhausted internal calculation that engineers do when they realize that the only honest path forward involves either telling the business something the business does not want to hear, or shipping something the engineers do not want to ship.

"We have to cut widgets," he said, eventually.

"From nine to —"

"Three. We can make three widgets fast. We cannot make nine widgets fast in ten days."

"Sales is going to lose his mind."

"Sales is going to lose his mind because he sold nine widgets that were never, technically, three weeks of work. Sales is going to lose his mind because *we* let him sell something that didn't exist."

"We."

He looked at me. There was no malice in it. He looked tired in a way I had never seen him look tired before.

"All of us," he said. "Me. You. The PM. The Sales Manager. The whole — the whole *thing.* We let the distance between the demo and the product get this big. We let the gap between *pretend.json* and a million users get this big. None of us closed it. None of us was brave enough to close it."

He was right. He was not, in my opinion, the most responsible person at the table for this particular failure. But he was also not the *least*. None of us were the least. There is no least, in a team of eight that has spent six months agreeing to be polite about the things that mattered.

I thought, that night, walking home, about a phrase the QA Tester had used in 3.1. *It is common, not normal.* I thought about how every single sub-chapter of QA, on this project, had been a different shape of the same lesson. Tests. Manual. Automation. Security. Performance. Each one a place where, if we had paid early, we would have paid less.

We had not paid early. We had paid never. Now the bill was due in ten days. The bill, as it always does, had interest.

We cut six widgets. We told the Sales Manager. The Sales Manager did, in fact, lose his mind. He lost it for forty minutes in a conference room. Then he calmed down. Then he made the calls. The calls went, to his astonishment, *better* than he had expected. The clients were tired of being lied to. They were, in some small way, relieved that we had finally said the word *less* out loud.

We shipped three widgets. The three widgets were fast. The three widgets, in production, did what they were supposed to do.

The other six widgets are still on the roadmap. They are, the roadmap says, "scheduled for Q1." It is, as I write this, the Q1 in question.

We are not, yet, on track to ship them.

We are, though, going to test them with realistic data this time.

That is what we have learned. It is, perhaps, all we have learned. It is more than we knew this time last year.

— END OF 3.5 —

3.6 — ACCESSIBILITY TESTS (THE MIRROR)

> "Our 'Login' button was an unlabeled div with an onClick. A screen-reader user told us, in a support ticket, that he had spent forty minutes trying to find out how to sign in. He called our app, with extraordinary patience, 'a beautiful room with no door.' I have never recovered."
>
> — anonymous developer, forum

The accessibility audit had been scheduled for the week before launch. It had been scheduled, originally, for the week *after* launch — a "post-launch polish item," in the words of a slide nobody now wished to remember — but the QA Tester had, at some point, walked into the Tech Lead's one-on-one with a printout of the WCAG criteria and explained, calmly and at length, that we did not have the luxury of an after.

The auditor was a woman who had been doing accessibility consulting for fourteen years. She arrived on a Tuesday, remote, on a video call, with the slightly weary patience of a person who has spent her career walking into rooms full of well-meaning developers who have, at no point in their lives, attempted to use a screen reader.

She shared her screen. She had pulled up our staging build. She had, the night before, run an automated audit. The audit had returned, she said, "a number." She did not, at first, give us the number. She walked us through the findings instead.

"Let's start with color contrast," she said.

She pulled up the dashboard. She put the cursor on the small grey timestamp text under each widget. The text was beautiful. It was, on the UI Designer's mood board, listed as *Whisper Grey*. It was, on a low-vision user's screen, indistinguishable from the background.

"This text," she said, "is at a contrast ratio of 1.9 to 1. WCAG AA requires 4.5 to 1 for body text. So this fails. This fails on every screen of your application that uses this color. By my count" — she paused, and clicked through a few screens — "this color appears on, conservatively, forty percent of the surfaces."

The UI Designer, who was on the call, said, "We — that color was a deliberate choice. It's meant to be subtle. For the secondary metadata."

"It is very subtle," the auditor said, kindly. "It is so subtle that a sighted user with normal vision can read it fine. It is so subtle that a sighted user with mild astigmatism, in the afternoon, after a long day, cannot. It is so subtle that a low-vision user — who is, I will remind everyone, a not insignificant portion of any user base — cannot read it at all."

She moved on. She moved on quickly, and she moved on a lot. The findings, by the time she had walked through them, were:

— Forty-eight elements failed contrast. — Twelve form fields had no associated labels. — The login button was a `<div>` with an `onClick`, not a `<button>`, meaning it was unreachable by keyboard and invisible to screen readers. — The dashboard widgets were rendered as `<div>`s with no semantic role, meaning a screen reader user heard, in sequence, "graphic, graphic, graphic, graphic, graphic," with no labels. — The export modal trapped focus incorrectly, stranding keyboard users on a button they could not escape from. — The error toast did not announce itself, meaning a screen reader user submitting a broken form received no feedback that the form had broken. — The mobile app's primary navigation was an icon with no accessible label, meaning users with screen readers heard, in place of "menu," only the word "button."

She paused. She let the list sit.

"And the number," she said, finally, "is two hundred and forty-one issues. That is the automated count. The manual audit will find more."

I want to say that I felt, in that moment, an appropriate remorse. I did not. I felt, instead, the small, hot, cold thing that I had been carrying around with me for weeks without naming. I felt the shape of it become clear.

"What does the auditor recommend," the Tech Lead said. He sounded, I noted, not quite sure he was the one speaking.

"Realistically?" she said. "You are not going to fix two hundred and forty-one issues in seven days. You can fix the legal blockers. You should fix the legal blockers. The button is a div. The labels are missing. Focus is trapped. Those are not aesthetic issues. Those are exclusion. You are, with the current build, shipping a product that a portion of your user base cannot use. Some of those users are protected by law. Some of them are not. All of them are, regardless of the law, customers who paid for a product they cannot open."

The UI Designer said, in the voice of a man being slowly crushed by a piece of furniture he himself had chosen: "We were going to do an accessibility pass after launch. It was on the polish list."

"There is never a polish phase," the QA Tester said, without looking up.

The auditor smiled, very slightly.

"Here is what I will tell you," she said. "Accessibility treated as a polish item is not accessibility. It is an apology. Real accessibility is decided in the design system, in the component library, in the ticket acceptance criteria. It is decided in the questions you ask in the design review. The fact that this is a two-hundred-and-forty-one-issue conversation, a week before launch, is not a failure of any one of you. It is a failure of the *moment* in the process at which you chose to begin asking the question. You asked too late. The lesson, for next time, is to ask earlier. The work, for this time, is to fix what you can in the time you have, and to be honest with your users about what you could not fix in time."

She let that sit.

"I will send the report by end of day," she said. "I will flag the legal blockers in red. I am happy to come back in a month and re-audit. I would like to encourage you, strongly, to invite me back."

She left the call.

We sat there. None of us said anything.

I want to tell you, in all honesty, what was happening inside me at that moment, because if I am writing any of this down at all, it is for this moment. Everything else was, in the end, plot.

I was thinking about a particular client.

The Sales Manager, on a call I had been pulled into in month two, had described to us our second pilot client. He had, in the way he described all clients, made them sound expensive and slightly mythical. He had said, in passing, that the main contact at that client — the one who would be using our portal every day, who would be demoing it to their executives, whose feedback would make or break the relationship — had a particular need. She was a woman who had had, since her thirties, a degenerative eye condition. She used a screen reader for about half of her work. She had told the Sales Manager, in a discovery call, that she had been excited about our product specifically because we had told her — we had told her — that we cared about accessibility.

We had told her this. We had told her this in March. The Sales Manager had told her this. The deck had said it. I had nodded along on the call. I had said nothing.

I had, the day before that call, opened a ticket in my backlog called *backend ARIA support — propagate label metadata through the API*. I had marked it *low priority*. I had said to myself, *I'll do it tomorrow*. I had deprioritized it, six times, over the next four months, in favor of more interesting work. The ticket had, by the time of the audit, been closed automatically by the JIRA bot for inactivity. I had not even bothered to re-open it.

The login button was a div because I had not propagated the label metadata. The dashboard widgets were unlabeled because I had not exposed the semantic information through the API. The error toast did not announce itself because I had not added the ARIA hooks to the response contract.

This was not a UI problem. This was not a design problem. This was not a "we got to it too late" problem. This was a *me* problem. I had had the responsibility. I had had the ticket. I had had the time. I had said *tomorrow*, and *tomorrow*, and *tomorrow*, and now a particular woman, at a particular client, was about to open our portal in seven days and discover that the company that had told her it cared about her had, in the very specific way of people who say they care and then do not show up, lied.

I sat at my desk for a long time after the meeting.

I opened my notebook. I went to the back. I went to the page I had been keeping, in small handwriting, since the kickoff. The page was titled *Things The Team Has Put Off*. It had, by then, forty-one entries. Each entry was in someone else's name. The UX Designer's research write- up. The UI Designer's tokens. The Tech Lead's PR queue. The Product Manager's UAT plan. Developer 1's API contract. Developer 3's certificate. The DevOps Engineer's test framework. The Sales Manager's penalty-clause math.

I had been keeping a ledger. I had been, in my own careful, polite, well-organized way, taking attendance at the failure of every single person around me. I had not, in five months, written down a single one of the things *I* had put off.

I turned the page. I started a new one. I titled it, slowly, in the same small handwriting:

Things I Have Put Off.

I wrote:

— the architecture doc. — the streaming dependency list. — the API contract for the frontend. — the test coverage on the consumer. — the SQL injection that I saw in review and let go. — the connection-reset TODO that is still in main. — the database optimization I warned about and then did not push for. — the ARIA support ticket I let JIRA auto-close. — every Slack message I drafted and did not send. — every PR I approved with the comment "typo." — every "I'll do it tomorrow" I said out loud. — every "I'll do it tomorrow" I said in my head.

I read the page. I read it three times.

I sat with the truth I had been working so hard not to look at.

The team was failing. That was true.

The team was failing partly *because of me*. Not as the hero who had been let down by lazy colleagues. As a *participant*. As one of the hands. As a quietly, respectably, well-mannered contributor to the slow unmaking of a thing I had told myself I cared about.

I read once, on a developer forum, in a thread about mid-career burnout, a comment that had, at the time, struck me as too clean to be real:

you spend the first half of your career learning to write code that works. you spend the second half learning that the code was never the problem. the problem was always the moments you did not speak up, the doc you did not write, the meeting you did not call, the colleague you did not push back on, the truth you did not say out loud. when you finally understand this, you stop being a developer in the old sense and you become, against your will, something else.

I had read the comment, the first time, with a small sneer. I read it now, in my head, with a steadier voice.

I closed the notebook. I closed my laptop. I stood up.

The QA Tester was at her desk. She was eating crackers from a packet.

I walked over.

"I need," I said, "to talk to you about something. Maybe tomorrow."

She looked up. She raised an eyebrow.

"Why tomorrow," she said.

I thought about it.

"You're right," I said. "Now."

She put down the crackers.

"Go on."

"I think," I said, slowly, because I had not said this to anyone yet, including myself, "I think I need to stop writing code on this project. I think I need to take over the project. I think someone has to do it, and I think it has to be me, and I think it has to start tomorrow, which means it has to start now."

She did not say anything for a moment.

Then she said, "Okay."

She said it the way you say *okay* when you have been waiting for someone to catch up to a thing you have known for a long time, and they finally have, and you do not intend to be cruel about how late they were.

"Okay," she said again. "What do you need from me."

I told her.

We started.

— END OF CHAPTER 3 —

Chapter 4 — Releasing the Feature

4.1 — CI / CD

> "The deploy script was 800 lines of bash, on one engineer's laptop. He was on vacation. His laptop was at the office. The office was closed for a public holiday. We waited three days to ship a one-line fix."
>
> — anonymous SRE, postmortem

The morning after I told the QA Tester I was taking over the project, I closed my IDE and did not open it again for eleven days.

This was the hardest part. The hardest part of becoming the project manager, when you have been a developer for fifteen years, is the part where you have to stop solving the problem yourself. It is the part where you have to look at a broken piece of code and *not* fix it, because if you fix it you will not, that day, write the deployment runbook, or call the clients, or sit with the DevOps Engineer and the QA Tester and triage what we are actually going to ship.

I closed the IDE on a Wednesday morning. I opened, in its place, a single Google Doc. I named the doc *Overmorrow — Path to Beta*. I wrote, at the top:

owners, dates, blockers. that is it. no more "soon." no more "tomorrow." no more "fine."

I shared the doc with the team. I scheduled a forty-five minute meeting for that afternoon. I titled the meeting, without irony: *who is doing what, by when, and what will stop them.*

The meeting was the most awkward forty-five minutes of my professional life up to that point. People did not, at first, understand why I was running it. The Tech Lead kept looking at me with the polite confusion of a man whose toaster has suddenly begun giving him career advice. The Product Manager joined late, from an airport, and said, "Hey, sorry, what are we — is this a pre-mortem? I love a pre-mortem." It was not a pre-mortem. It was, in fact, a mortem. I did not correct him.

I went around the room. I asked each person, one by one, the same three questions.

1. What are you actually working on this week? 2. What date will it be done? 3. What is going to stop it from being done by that date?

The first time I asked these questions, I got, in the round, the following answers, which I am paraphrasing only slightly:

Developer 1 — "various things, by, you know, soon, no blockers." Tech Lead — "code review, generally, ongoing, no hard date." DevOps — "trying to make the build green, by Friday probably, blocker is everyone." QA Tester — "manual regression, by launch day, blocker is the bug count." Sales Mgr — "client communications, ongoing, blocker is what to actually tell them." Product Mgr — "I am, between meetings, you know, keeping things moving."

I let the answers sit for a moment. Then I said, "Okay. We are going to redo this. Each of you, again. With *one* piece of work. With a *date*. And the blocker has to be a specific thing a specific person can specifically remove. Not 'everyone.' Not 'the bug count.' A name. A thing. A day."

The Tech Lead, slowly, said, "This is — a different way of running this meeting."

"Yes."

"Okay."

We went around again. It took twenty minutes. By the end, the doc had concrete entries. Developer 1 owned the API client refactor, due Friday, blocked by me approving her PR. The Tech Lead owned the migration script, due Monday, blocked by nothing, which is to say, blocked by the absence of someone making him do it. DevOps owned the production deploy pipeline, due Wednesday, blocked by an infrastructure approval he had been waiting on for ten days from a person whose name we wrote down so I could chase them personally that afternoon.

The Sales Manager owned the client communication, due Thursday, blocked by *me*, because he needed me to write the technical paragraph of the delay email.

I wrote the technical paragraph that night. I rewrote it six times. The Sales Manager and I then rewrote it seven more times together. It went out on Thursday morning, to all three pilot clients, with the subject line *Beta launch update — please read*. It said, in clear, plain words:

We are moving the beta by ten business days. The reason is that, in our final pre-launch testing, we identified three classes of issues — security, performance, accessibility — that we are not willing to ship past. We will use the next ten business days to remediate them, and we will hold a joint go/no-go call with each of you on the new launch date. We are sorry. We should have flagged this earlier. The fault is ours.

We waited, that morning, with a kind of held breath, for the replies. The first reply came in twenty-three minutes. It was from the largest client. It said:

Thank you for telling us. Most vendors find a way to dress this up. We appreciate that you did not. See you in ten days.

I read that email three times. I sent it to the Sales Manager. He did not say anything. He just sent back a single thumbs-up. I think it was the only honest thumbs-up he had given in six months.

Meanwhile, the CI/CD work was its own small siege.

The DevOps Engineer and I sat together, in the corner of the office, for three nights in a row. We did, on those three nights, the work that should have been done in month one. We deleted the old Jenkins job. We replaced it with a real pipeline. We added the smoke tests the QA Tester had written. We added the deployment runbook. We documented the rollback

procedure. We rotated all the hardcoded credentials. We moved the production secrets into the actual secrets manager — which had, for six months, been provisioned and empty, because nobody had ever migrated to it. We added a deploy gate that required two human approvals before any push to production. We added a smoke check that ran post-deploy and rolled back automatically if the dashboard error rate spiked.

It was, in total, about three weeks of work compressed into about sixty hours. We did it. We did it the way people do everything in the last week of a doomed project: by drinking too much coffee, by eating too much pizza, by making decisions slightly too quickly, by trusting each other in ways that, on a less desperate timeline, we would not have trusted each other.

On the second night, around two a.m., the DevOps Engineer turned to me. He had not shaved in some time.

"Why are you doing this," he said.

"Doing what."

"This. The PM thing. You don't have to. Nobody asked you to. The Product Manager is, technically, the Product Manager. You could just — keep coding. It is not, in any formal sense, your job."

I thought about this for a long moment.

"I read something once," I said. "On a developer forum. A guy said: *the senior engineer's most important deliverable is the meeting they ran instead of the code they wrote.* I always thought that was bullshit. Like, deeply. I thought it was the kind of thing managers tell engineers to make engineers feel good about being slowly turned into managers. I was annoyed by it for years."

"And now?"

"Now I think it was the most true thing I ever read about this job, and I refused to look at it because looking at it would mean admitting that the meeting I had not run was a meeting that, on this project, no one else was going to run, and that the thing I was protecting by writing code instead was, in fact, just my own preference for code over people."

He looked at me. He said, "That is bleak as hell."

"It is."

"Cool," he said. "Pass me the laptop. I want to write this deploy hook before either of us thinks too hard."

We shipped, on the eleventh day, a beta. It was a beta. It was not the beta we had promised in March. It had three widgets, not nine. It had a manual export instead of an automated one. It had no mobile app — Developer 3 had, with my full and explicit blessing, descoped the mobile app to a follow-up release, with a timeline we wrote down and posted publicly. It had,

in place of nine widgets, the most beautiful CI pipeline I have ever personally been associated with, complete with a green checkmark that, this time, I trusted.

I trusted it because I had watched, with my own eyes, the DevOps Engineer add a real test that failed for a real reason and then a real engineer fix it. Not a placeholder. Not a Playwright homepage. A real test, a real fix, a real deploy. We had, in three weeks of triage, more genuine test coverage than we had had in six months of building.

The clients logged in. The dashboard loaded in 1.4 seconds. The login worked. The export worked. The error toast announced itself to a screen reader. The login button was, finally, a button.

I read once, on a developer forum, in a thread titled *the worst launch you ever shipped*, a comment that said:

the launches I am proudest of are not the ones with the longest feature lists. they are the ones where i can look at every single thing in the build and say "this works, on purpose, because someone decided it would." most of my career, that has not been the case. i don't know what to call most of my career.

I thought about that comment a lot in the days after we shipped. I think I now know, slightly better, what to call most of my career so far.

I am trying to call the next part something different.

— END OF 4.1 —

4.2 — DOCUMENTATION

"The only architecture document was a Confluence page that said 'TODO,' last edited four years ago, by a person who no longer worked at the company. We treated it, eventually, as a kind of sacred relic. We were afraid to delete it because it was the only thing we had."

— anonymous engineer, internal wiki rant

The architecture document I had been promising to write since week one of the project was, on the morning of the new go-live, still the same six-line stub I had created six months earlier:

Project Overmorrow — Architecture # Author: me # Status: DRAFT # Last updated: [the date, six months ago]

1. Overview The Customer Portal is composed of: services

That was it. *Services.* A noun followed by nothing.

I would like to tell you that I sat down, the day after the beta shipped, and wrote the architecture document in a single virtuous afternoon. I would like to tell you it unspooled out of me cleanly, and that I leaned back in my chair when it was done and felt the warm satisfaction of the long-postponed task at last completed.

I would like to tell you that, but it would be a lie. The architecture document, like all real documents, was written the way real documents are written, which is to say: badly, in pieces, over the course of three weeks, in the margins of meetings and the small spaces between fires, with the help of approximately every other person on the team and a small but persistent feeling of nausea.

I started by sending a message in the team channel:

@channel — i am writing the architecture doc this week. i need 30 minutes from each of you. i will come to your desk. i will ask you questions. you will answer them. that is the whole protocol.

There was a small silence. Then Developer 1 replied:

DEV 1: do i have to draw anything

I said no. I said she just had to talk to me. She replied with a single sad emoji and then, ten minutes later, *ok fine, after lunch*. This was, by Developer 1's standards, enthusiasm.

I went, that week, to every desk. I sat with each person. I asked each of them, in their own area, the same five questions:

1. What is the thing you own? 2. What does it depend on? 3. What depends on it? 4. How does someone, six months from now, who has never seen this code, change it without breaking something? 5. What is the one thing that, if you got hit by a bus tomorrow, would take the team three days to figure out?

The fifth question was, I will tell you, the most useful question I have ever asked anyone in a software company. It works because it bypasses the human urge to be modest and the human urge to be heroic at the same time. People will not tell you what is fragile. They will tell you, if you ask them carefully, what would be hard to replace *them* on. The two answers are, almost always, the same answer.

The list of *bus answers* I gathered that week was a small horror.

Tech Lead — "the auth claims map. it's mostly in my head. it's also written in a way that nobody else can read. there's a comment at the top that says 'do not touch.' that comment is from me. to myself."

Dev 1 — "the design tokens. they're hex codes in a file. nobody knows which token in figma maps to which hex. only i know. and i don't know all of them."

Dev 3 — "the iOS provisioning. it's tied to my personal apple id because we never set up a corporate one. if i leave, the next mobile dev will have to re-create everything from scratch. this is a known issue. it has been known for a year."

DevOps — "the deploy pipeline secrets. they're in a vault i set up. only i have the master credential. if i go, you'll have to break the glass on a process that has, technically, never been tested."

QA Tester — "the test plan spreadsheet. it lives in my personal drive. there is no backup. there is no version history beyond what google docs gives you. it has six months of context in it that has never been written down anywhere else."

Me — "the streaming consumer. the wrapper around the legacy framework. i wrote it in a panic in week three. there is no documentation. there are, now, finally, tests, but the *why* of the design is in nobody's head except mine."

I read the list back to myself. I read it twice. I felt, for a moment, an almost physical understanding of what an *organization* is. An organization is a collection of small private rooms, in the heads of individual people, each of which contains some critical piece of how the thing actually works, and most of which have no key. Documentation is, in this metaphor, the act of cutting keys. It is unglamorous. It is finicky. It is, generally, boring. It is, also, the only thing that lets the organization continue to exist without you.

I wrote the architecture doc that week. It was eighteen pages. It had, in it:

— A diagram. A real diagram. With arrows that actually connected to things. Drawn by me, in a tool, with rectangles that had labels, and lines that had directions, and a legend that said

what the colors meant. It took me four hours to draw, across two evenings. It was, in retrospect, four of the most useful hours I spent in my entire time on the project.

— A list of every service. With a one-paragraph description. With its owner. With its dependencies. With its known limitations. The known limitations section was, on most services, longer than the description. This was, I decided, a feature, not a bug.

— A section called *Why We Built It This Way*. This section was, by far, the most painful to write. I had to call out, in writing, the specific decisions we had made and explain why. I had to write, in plain English, the sentence: *We chose to wrap the legacy framework for streaming, even though it does not natively support streaming, because at the time the team had no bandwidth to adopt a new framework. This decision is paying interest in the form of [the next paragraph] and should be revisited in [date].* I wrote that sentence with my own hand. I wrote it because I knew, with a clarity that only comes from having been the person who built the thing, that if I did not write it, someone in two years would inherit the wrapper, hate me for it, and then make the exact same wrapper, again, with slightly different naming.

— A section called *Things That Will Bite Us Eventually*. A list of every TODO that was still in the code, every "we'll fix it in the polish phase" that had not, in fact, been fixed, every credential that needed rotation in ninety days, every dependency that was approaching end of life. The section had thirty-two items. I gave each of them an owner and a date. I assigned some of them to myself.

— A section called *How To Work On This Codebase When You Are New*. A short, ruthless guide. Where to clone the repo. Which env vars you need. Which ones to skip. Where the runbooks live. Who to ping in which Slack channel. The fact that the channel called #urgent was, despite its name, mostly used for cat photos, and that the channel called #random was where the actual urgent things happened. The cultural map matters more than the technical one. It always does. I wish I had been told this as a junior.

I shipped the doc on a Friday afternoon. I put it in the team channel with the message:

the architecture doc is done. please read it. it will be wrong in places. tell me where. we will update it. it lives at [link]. it is no longer a sacred relic. it is a working document. treat it accordingly.

The Tech Lead read it that weekend. He came to my desk on Monday. He had marked it up. There were, he said, twelve things wrong. He had a list. He went through them with me. They were, in fact, twelve real things, and we fixed them together that morning. By lunchtime, the document had a real version-two stamp on it, and a real edit history with two real authors, and a real list of known limitations agreed by both of us.

He stood up to leave. He stopped at the door of the huddle room.

"Hey," he said. "I should have written this."

"You wouldn't have."

"I know."

"It's fine."

"It isn't fine. But thanks."

I read once, on a developer forum, in a long thread about documentation, a comment that I have been trying, since, to live by:

the documentation that does not exist is not documentation that has not been written. it is documentation that has been written by absence. it teaches every new engineer the same lesson, in the same painful way, and the lesson it teaches is always: *you are alone here.* writing real documentation is the act of revoking that lesson.

I thought about that line, that Monday lunchtime, walking back from the kitchen with a fresh coffee, with the architecture doc finally, six months late, in version two on the team wiki, with a real diagram and a real list of known fragilities and a real instruction for whoever came next.

I thought, with a small, slightly tired pride: *the next person is not alone here.*

It was not a feeling I had had often, on this project.

It was the best one I had had so far.

— END OF CHAPTER 4 —

Chapter 5 — Maintenance

5.1 — CUSTOMER SUPPORT

"I had been on the support team for six months when I learned, from an angry customer, that we had launched a new feature. The customer was reading from the press release. I had not seen the press release. The press release had been written by a marketing team I had never met. The feature did not work. I apologized for the feature for ninety more minutes. The customer hung up on me. I do not blame the customer."

— anonymous support agent, blog post

We met the Customer Support Rep on the second day after launch.

She had been hired, three months earlier, by a different part of the org. We had not, in any meaningful sense, been told this. She had been told, in turn, that she was joining "the customer team for a new product launch." The new product, in her onboarding documents, had been described in two sentences. The two sentences were, on re-reading them later, both technically true and substantively useless.

She had, on day one of her job, sent a message to a distribution list called *@product-team*. The message had said, politely, *Hi all — I'm the new support rep for the upcoming portal launch. Could someone walk me through the product? Looking forward to working with you.* The distribution list had, it turned out, been set up to forward to a Slack channel that the team had muted, because the Sales Manager had, one boring afternoon, used it to send seventeen polls about which sandwich place to order from. Nobody on the engineering team had checked the channel in six weeks. The Customer Support Rep's onboarding message had sat there, unread, for ninety-one days.

She had filled the ninety-one days the way support people fill them when the team they are supporting is silent. She had read the public-facing marketing materials. She had clicked around the demo environment. She had, with the admirable initiative of a person whose job did not yet formally exist, written her own draft FAQ based entirely on guesswork. She had answered, in her first month, six support tickets. Of those six, she later told me, one had been a customer asking when the product would launch, and she had, with the polite uncertainty of a person who had not been told, written back *we don't have a confirmed date yet, but I will follow up as soon as I do.* The customer had then forwarded the email to their account manager, who had forwarded it to the Sales Manager, who had, in a blistering eight-paragraph reply, demanded to know which of the engineering managers was *leaking launch dates* to support. There had been no leaked launch date. The Customer Support Rep had simply not had one to share.

This had been her welcome to the team. By the time we launched, she had been working for the company for almost four months and had spoken, by her own count, to exactly one engineer, briefly, in an elevator, about the printer.

I learned all of this on a Tuesday morning, the second day after launch, when she walked into the engineering area for the first time. She was holding a laptop. She had, on her face, the particular composed fury of a person who has just spent forty consecutive minutes on a support call with a paying customer trying to explain a feature she had never been told existed. She put the laptop down on my desk.

"Are you," she said, with a flatness that I admired immediately, "the project manager."

I had, by that morning, been the project manager for roughly eighteen days.

"Yes," I said.

"My name is" — and she said her name, but I am keeping the convention of this story, so I will call her, as this story has, the Customer Support Rep — "and I would like, very politely, to know why I have spent the last forty minutes trying to walk a customer through your export feature with no documentation, no demo account, no training, no FAQ, and no idea that the export feature even existed until this morning when the customer called in to report that it was broken."

I had no answer for any of this. I had, in fact, all of the answers, but each of them was a confession, and I was beginning to learn that one of the things being a project manager required was to receive other people's fair anger without immediately deflecting it.

"That is a fair question," I said. "I do not have a good answer. I am sorry. Will you sit with me for an hour and tell me everything you needed to know two months ago?"

She blinked. I think she had been prepared for a fight.

"Yes," she said, eventually. "I will."

We sat for two hours. I cancelled three other meetings. She walked me through every ticket she had received in the last forty-eight hours, in detail. There were, by her count, sixty-three tickets. She had categorized them, with the natural taxonomic instinct of a good support person, into four buckets:

— *Things I should have known.* (The product exists. The product launched on Monday. The export feature is asynchronous and can take up to ten minutes for large datasets. The mobile app does not yet exist.) Twenty-one tickets.

— *Things you should have told the customer.* (We have only three dashboard widgets, not nine. The accessibility issues are being patched in the next release. Single sign-on is supported only for two of the three identity providers we previously listed.) Eighteen tickets.

— *Things that are actually broken.* (Login fails for users with apostrophes in their names. The dashboard widget for billing shows yesterday's data, not today's. The export job, when it finishes, sends the email to the wrong address if the user has updated their profile in the last twenty-four hours.) Fifteen tickets.

— *Things I cannot triage because I do not know enough to triage them.* Nine tickets. She read one of them out loud. It was a customer describing an interaction with our system in a level of detail that suggested they were both extremely competent and extremely frustrated. The Customer Support Rep had, on the call, escalated it to "engineering." She had not known who in engineering to send it to. She had sent it to the generic engineering distribution list. The list had, of course, been the same muted Slack channel as her onboarding message. The ticket had, accordingly, sat unread for eight hours.

I sat with this. I sat with the size of it. Sixty-three tickets, in forty-eight hours, from one human being, with no support beyond what she had built for herself, in a job she had been told was "for a new product launch" and about which she had been told approximately nothing.

"Okay," I said. "Here is what we are going to do. Today. Now. You and I. I am cancelling the rest of my meetings."

"Okay."

"We are going to write the things you should have known. A real product overview. A real feature list. A real list of what we shipped, what we did not ship, and when the things we did not ship will arrive. Plain language. One page. By end of day."

"Okay."

"We are going to write a triage matrix. A real one. Which issues go to which engineer. With names. With backup names. With response time expectations. By Friday."

"Okay."

"We are going to set up a daily fifteen-minute sync between support and engineering. Every morning. Nine-fifteen. I will personally be in it. I will personally make the Tech Lead and at least one developer be in it. We will go through your top five tickets. Every day. Without exception."

"Okay."

"And I am going to apologize, on behalf of the entire engineering team, for the fact that you have done four months of this job alone, and that the first time anyone from engineering sat down with you was today, after you had to walk over and demand it. That was wrong. I was part of that wrongness. I am sorry."

She looked at me for a long moment.

"Why did you do that," she said.

"Apologize?"

"Yes."

"Because it was true."

She nodded, slowly. She did not smile. But she relaxed, slightly, in the way a person relaxes when they have been carrying something they did not realize they had been carrying alone.

"Okay," she said. "Then let's start."

We started.

By the end of that week, the support team — which had been, until the Tuesday, one person — had a real product overview, a real triage matrix, a real escalation path, a real backup, a real seat at the daily standup, and a real engineering counterpart on speed dial. The number of tickets did not decrease that week. It actually went up, because the support rep was now better-equipped to classify them, and so she was finding, in the pile, more real bugs. But the *time-to-resolution* dropped, by my rough count, by half. The customers stopped, mostly, yelling. The customers, in some cases, even started to be patient with us. This was, I will tell you, not a thing I had ever before observed customers do.

I read once, on a developer forum, in a long sad thread about engineering-support relations, the comment:

every product team has, somewhere in its building, a person who knows what the customers actually think of the product. that person is in the support queue. they have always been in the support queue. you have never once asked them. they have been waiting, for the entire life of the product, for someone from engineering to walk over and say hello. when you finally do, you will discover that they have been quietly running half the company without you. it will be the most humbling conversation of your career.

I had not, until that Tuesday, walked over.

I have, since then, walked over a great many times.

I do not, now, understand how I ever thought it was optional.

— END OF 5.1 —

5.2 — MONITORING

> "We learned about the outage from a tweet. The tweet was from a competitor. The tweet was a screenshot of our status page. Our status page said 'all systems operational.' Our systems were not operational. Our status page was an HTML file someone had copied off the internet in 2019."
>
> — anonymous SRE, conference talk

The thing about monitoring is that it works perfectly until the very moment you most need it, and then it does not work at all. This is not because monitoring tools are bad. It is because, on most teams, monitoring is a thing somebody set up once, several years ago, on a Wednesday afternoon, in a hurry, before going on vacation, and then never touched again.

Our monitoring, in week one of being live, fit this description with embarrassing precision.

It had been set up by the DevOps Engineer, in fairness to him, in his first sprint on the project, before any of the hard work had landed on him. He had stood up a dashboard. The dashboard had four panels. The panels were:

— request rate — error rate — average response time — number of currently logged-in users

He had, on the same afternoon, set up three alerts:

— error rate > 5% for 5 minutes — page on-call — response time > 2 seconds for 10 minutes — page on-call — zero requests for 5 minutes — page on-call

These alerts had been configured against his personal phone. They had been configured against his personal phone because the team had not, at that time, had an on-call rotation. The on-call rotation was, like the test framework and the architecture doc and several other things, on a list of *items to set up properly later*. The list had grown, the way these lists grow, until *later* became *whenever the next disaster forces us to*.

Now we had launched. We had, in the kindly euphemism of our internal status report, *gone live*. And the monitoring, such as it was, was paging the DevOps Engineer's personal phone, twenty-four hours a day, every day, for any of three thresholds he had picked in fifteen minutes of work seven months earlier.

I learned this on the third day after launch. I learned it because the DevOps Engineer arrived at the morning standup looking like a man who had been visited, in his sleep, by ghosts he had personally designed.

"How are you," I asked.

"I have," he said, very quietly, "received forty-seven pages in the last seventy-two hours."

"Forty-seven."

"Forty-seven."

"Were any of them real."

He considered. He considered for, perhaps, ten seconds. This was, in itself, an answer.

"Three," he said, finally. "Three were real. The other forty-four were the dashboard noticing, accurately, that something was happening, and then, inaccurately, classifying it as an emergency. One of the alerts," he added, with the dead-eyed amusement of a man who had not slept, "was triggered, at three a.m., because the response time on the static-asset endpoint went from forty milliseconds to ninety milliseconds, which crossed the *2 seconds* threshold by a factor of, mathematically, not at all. The alert fired anyway. I do not know why. The alert configuration is the alert configuration. The alert configuration was made by me, in 2024, with my own hands. I am, at this moment, both the perpetrator and the victim of the same crime."

I cancelled my morning. We sat down together. We did, that day, and the next two days, the work that should have been ongoing for the entire life of the project. We did, in those three days, a small and unglamorous thing: we made the monitoring *match the system*.

We deleted the four-panel dashboard. We replaced it with twelve panels, each one tied to a real user-facing behavior:

— *can a user log in* (synthetic check, every minute) — *does the dashboard load* (synthetic check, every minute) — *does the export work* (synthetic check, every five minutes) — error rate, by endpoint, broken out — p95 latency, by endpoint, broken out — database connection pool saturation — kafka consumer lag — auth token issuance rate (catches our login working, but our session not establishing — the OVR-204 class of bug) — bytes ingested into the streaming layer — open support tickets, by category, surfaced from the support tool's API — deploy markers, so we could see, on the same timeline, when each release happened — and, last, a panel labeled *manual feel* — a free slot for whatever the on-call had been worrying about that morning

We threw out the alerts. We rewrote them, from scratch. Each new alert had to pass three tests, which we wrote on a sticky note and stuck to the DevOps Engineer's monitor:

1. is this paging an actual human? 2. does the actual human know what to do when paged? 3. is the actual human's runbook for this in a place the actual human can find at three a.m. with one hand on the bedside lamp and the other hand on the laptop?

If the answer to all three was *yes*, we kept the alert. If the answer to any was *no*, we either deleted the alert or wrote the runbook. We wrote, that week, seventeen runbooks. They were short. They were ugly. They were, in many cases, simply the words *call X. ask them about Y. do not, under any circumstances, do Z.* They were, despite their ugliness, the most useful piece of writing the team produced that month. By the end of the second week post-launch, every alert had a runbook, and every runbook had a name on it, and the names had agreed in advance to the responsibility.

We set up an on-call rotation. A real one. Three engineers, including me, rotating week-on/week-off. One weekend each every six weeks. We documented the handover. We documented the escalation. We documented, because we had learned the hard way, the line at which on-call should *stop trying to fix it and call someone who can actually decide things*. We added that line in boldface, in the runbook, after a Friday night when the DevOps Engineer, alone, had spent four hours trying to patch a problem that should have been escalated to me, and had not been escalated, because nothing in the runbook had told him he was allowed to wake me up.

The first week of the new monitoring, our pages dropped from fourteen a day to two a day. The two a day were real. The two a day were caught. The two a day were fixed before customers, in many cases, even noticed. The DevOps Engineer began to look, in standups, like a man who had recently rediscovered the concept of breakfast.

There was, on the second Friday of the new system, a real incident. A real one. The kind that would have, two months earlier, taken us six hours to find.

The synthetic check on *does the dashboard load* failed, quietly, at 2:14 p.m. The page went out — to me, that week, on rotation. The runbook, which I had written myself ten days earlier, said:

1. confirm the failure is reproducible (open the app in a private browser window). 2. check the dashboard error rate panel. 3. check the streaming consumer lag panel. 4. check the database CPU panel. 5. if (3) is rising sharply and (4) is normal, the consumer is wedged. restart the consumer pod. 6. if (3) is normal and (4) is rising, the query is hot. flush the query cache. 7. if neither of the above, escalate.

I followed the runbook. (3) was rising. (4) was normal. I restarted the consumer pod. The dashboard came back at 2:21 p.m. The total customer-facing impact was seven minutes. I went back to my meeting. I sent a message in the team channel:

consumer pod wedged at 14:14. restarted via runbook. recovered at 14:21. total user impact: 7 min. root cause investigation: post-mortem ticket OVR-1031.

That, I will tell you, was the proudest message I have ever sent in a team chat. Not because the consumer pod had wedged. Consumer pods wedge. Software is software. But because, this time, the system had told us. The system had told us *the right thing*, in *the right voice*, to *the right person*, with *the right instructions*. The whole apparatus had worked. Seven minutes was, in production-software terms, a rounding error. It would, two months earlier, have been six hours.

I read once, on a developer forum, in a thread on incident response, a comment that has since become a small private creed:

you cannot prevent every outage. you can only decide, in advance, whether you want to find out about your outages from your monitoring or from your customers. one of those two will

tell you first. the choice is, mostly, made in the months before anything bad happens. it is rarely made on the night.

We were, finally, finding out from our monitoring.

It had only taken us six and a half months, an unscheduled launch delay, three nights with the DevOps Engineer on a couch eating cold pizza, and the public humiliation of a one-star App Store review that had simply read *the app is dead, did your company also die*, to learn this lesson.

We did, eventually, learn it.

It is the only lesson, on the entire project, that I am absolutely sure has stuck.

— END OF CHAPTER 5 —

Chapter 6 — Promotion

6.1 — GOING LIVE (FOR REAL THIS TIME)

"On the morning of GA we did not eat. We did not speak above a whisper. We watched the dashboard the way other people watch ultrasounds. When the first real customer logged in, the on-call engineer wept, briefly, into a printout of the runbook. The PM took a photo. The PM is not allowed to share the photo. We have agreements about this."

— anonymous engineer, internal blog

The general availability launch — the *real* launch, the one with no asterisks and no carefully worded delay emails and no quiet client calls about which features had been descoped — was scheduled for a Tuesday morning in month seven.

We chose Tuesday because, as the DevOps Engineer had patiently explained in a meeting two weeks earlier, *no serious system goes live on a Friday, unless what you want, on the deepest level, is to ruin a weekend.* He had said this with the small dead smile of a man who had, in his career, had several weekends ruined by Friday launches, and who had decided, in his late thirties, to make his preferences known with his whole chest.

We agreed on Tuesday. We picked nine a.m. local time. We sent the customers calendar invites. The Sales Manager sent each of his three pilot clients, by hand, a personal note. The note said, in part:

We have shipped beta to you for the last six weeks. You have been generous and patient. On Tuesday, we flip the switch from beta to GA. Nothing about your experience will change. We just stop pretending it is provisional. Thank you for staying with us.

He showed me the note before sending it. It was the shortest customer email he had written in his career. I told him so. He said, "Yeah. I am, in my old age, beginning to suspect that the long ones were never read."

The morning of the launch, we were all in the office by seven-thirty. Nobody had asked anyone to come in early. Everyone had, independently, decided to. The Tech Lead had brought, without explanation, a box of pastries he had bought at a place that was both inconveniently far from the office and extremely good. The Customer Support Rep had brought, in a covered tin, what turned out to be homemade cookies. Developer 1 had brought, with grim determination, four laptops, three of which were not hers and which she had collected the previous evening from people she did not, technically, trust to be on time. The QA Tester had brought a printed checklist. The checklist had ninety-three items. She had, by the time the rest of us arrived, already crossed off seventeen of them.

The DevOps Engineer was at his usual corner. He had two monitors. The left monitor showed the new dashboard — the twelve-panel one we had built in the post-beta sprint, the one with synthetic checks and runbook links and deploy markers, the one that, every time I looked at

it, made me feel something I had not, on this project, felt before: *confidence.* The right monitor showed the deploy pipeline, which was sitting at a single button labeled *Promote to GA*. The button was grey. It was waiting on two human approvals.

I gave my approval at eight-fifty-three. The Tech Lead gave his at eight-fifty-five. The button turned green. The DevOps Engineer looked at it. He did not press it immediately. He did, instead, the thing he always did before any irreversible action: he leaned back in his chair, took a long breath, looked at the ceiling for exactly three seconds, and then leaned forward and clicked the button without ceremony.

The pipeline ran for eleven minutes. We did not talk for eleven minutes. We watched the steps go green, one by one. Build. Test. Image push. Canary. Health check. Rollout. Smoke. Each step took the time it took. Each step finished. There were no surprises. There was, in particular, no moment in which the pipeline turned red and sent the DevOps Engineer's monitor into the special panic shade of red that was, by team consensus, the worst shade of red in the office.

At nine-oh-four, the pipeline finished. The status panel turned green. The deploy marker dropped onto the dashboard graph. A small notification went out to the team channel:

DEPLOY: portal-prod 2.0.0-ga succeeded.

The Sales Manager, standing behind us, said, "Is — is that it. Is that the whole thing."

"That is the whole thing," the DevOps Engineer said.

"It's so quiet."

"Yes."

"In the demo videos, when companies launch things, there is, like — confetti. People hug. Champagne. There was a guy, in one video I saw, who rang a bell."

"We do not have a bell."

"Should we have a bell."

"No," the QA Tester said, without looking up from her checklist. "We should have a runbook. We have a runbook. That is the bell."

The Sales Manager nodded, slowly. He was, I could tell, a little disappointed by the absence of a bell.

At nine-oh-six, the first customer logged in. We watched the dashboard. The synthetic check on *can a user log in* went green at the bottom of the next minute. The dashboard latency stayed at 1.3 seconds at the median. The error rate, on every endpoint, sat at zero. The streaming consumer happily ingested the day's first batch of events. The export endpoint accepted a small request and returned a small file. The login button, somewhere out in the wide world, was a button.

By ten o'clock, four hundred and twelve customers had logged in. By noon, eleven hundred. By the end of the first day, three thousand and four. The dashboard latency held. The error rate stayed under half a percent. The support queue, which had been a small pile in beta, became a slightly larger pile, but the larger pile was the kind of pile a healthy product produces — *I cannot find the export button*, *can I add a teammate*, *how do I download my receipts* — not the kind of pile the broken project had produced — *the page does not load*, *I have been logged out for the eleventh time today*, *did your company also die*.

I read once, on a developer forum, in a thread titled *the most boring launch you ever had*, the comment:

the most boring launch is the best launch. the most boring launch is the one where everyone has already done their job, weeks earlier, in small unglamorous ways, so that when the day comes nothing happens to be done. boring launches are evidence that everything that needed to be anxious has already been anxious. the anxiety has been *paid down*. it does not, on the day, have to be felt again. people who chase the dramatic launch do not understand that they are chasing the symptoms of incomplete preparation.

We had, that Tuesday, the most boring launch of any of our careers.

It was, against all of my prior intuition, the proudest day of mine.

At lunch, we walked across the street together. All nine of us — eight engineers and the Customer Support Rep, who I had insisted come and who had, with some resistance from the Sales Manager about *whether support people typically attend launch lunches*, been extended a formal invitation by me, the project manager, in writing. We sat at one long table. We ate sandwiches. The Tech Lead, halfway through his sandwich, looked up and said, mildly:

"Is this — is this what it's supposed to feel like."

"What."

"This. The launch. The non-emergency. The lunch."

I thought about it.

"I don't know," I said. "But I think we should let it feel like this for the rest of the day. We can decide tomorrow whether it counts."

The Sales Manager raised his sandwich.

"To the dashboard," he said.

"To the dashboard," we said.

We touched our sandwiches together, gently. It was the most embarrassing toast any of us had ever made.

It was also, I will tell you, the most honest one.

— END OF 6.1 —

6.2 — THE RETROSPECTIVE

"At the end of the retro, the senior engineer raised his hand and said, 'I owe everyone here an apology.' The room went very quiet. Then the next engineer raised her hand. Then the next. The retrospective ran ninety minutes over. Nobody left. We learned more in those ninety minutes than in the previous two years."

— anonymous engineering manager, blog

I had, in my fifteen years as a developer, been to many retrospectives.

Most of them had been, charitably, theatre. Somebody would draw three columns on a whiteboard. *Went well. Could be better. Action items.* Somebody else would write, in the *went well* column, *team morale*. Somebody else would write, in the *could be better* column, *communication*. Nobody would write anything in the *action items* column, because nobody, including the person running the retrospective, expected the retrospective to produce action items. The action items were not the point. The point was that the company calendar had, scheduled on it, a thing called *retrospective*, and the company felt better about itself when that thing happened, and so we all sat in a room for an hour and pretended.

I had, when I became the project manager of this team eighteen days before launch, made the team a promise. I had said, in our first meeting, in that very awkward forty-five minutes where I had asked everyone what they were working on:

when we ship — and we *will* ship — we will hold a retrospective. it will not be theatre. it will not be three columns on a whiteboard. it will be a room where, for two hours, we tell each other the truth about the last six months. anybody who does not want to be in that room does not have to be in that room. but the room is going to happen. i am going to make sure of it.

I had, by Tuesday's launch, mostly forgotten this promise. The QA Tester, of course, had not.

She came to my desk on the Wednesday, the day after the quiet launch, the day when the support queue still held and the dashboard still showed under one percent errors and the team was, collectively, beginning to feel the specific kind of after-the-storm tiredness that comes when you realize the storm is, at last, behind you.

"You promised a retrospective," she said.

"I did."

"It is now."

"It is now?"

"Yes. Friday. Two hours. Big conference room. I have booked it. You are running it. I have already told the team."

I had, by then, learned that there was no point in arguing with the QA Tester about scheduled events. I spent Wednesday and Thursday writing prompts. Not three columns on a whiteboard. A small list of questions that each person would answer, in turn, around the table:

1. What did you do on this project that you are proud of? 2. What did you do on this project that you wish you had done differently? 3. What did *we* — as a team, as a system — do that you wish we had done differently? 4. What did you *not* do, that you should have, that you would like to confess to the room right now? 5. What is one thing you commit to doing differently on the next project?

I was most worried about question four. Question four was the dangerous one. Question four asked people to admit, out loud, in front of their colleagues, the specific thing they had not done. I thought, going into Friday, that question four would die. People would skip it. People would mumble. People would say, *I think I've covered that already*, and look at the wall.

I was wrong.

We sat in the big conference room at two p.m. on Friday. We had eight people around the table — the seven of us, plus the Customer Support Rep, who I had personally invited, and who had personally accepted, and who I had personally promised would not be the first person asked to answer the difficult questions. The Product Manager was on a video call from a hotel lobby in another city, his face slightly pixellated, but present, which was, in itself, the most consistent he had been on this project in months.

I went around the room. I asked the first three questions. The answers were the kind of answers you would expect. People were proud of specific shipped pieces. People wished they had pushed back harder on specific decisions. People had, generally, kind things to say about each other.

Then I got to question four.

I did not, this time, ask someone else to start. I had decided in advance that, if question four was going to work, I had to start it.

"I will go first," I said. "I am going to answer question four. Please listen."

I took a breath.

"I did not write the architecture document for six months. I had it as my responsibility from week one. I opened the file in week one. I wrote the title and one sentence. I did not touch it again until after we had already broken the things that the document would have prevented. I told myself, every week, that I would do it *tomorrow*. I did this for twenty-six tomorrows. Many of the worst things on this project — the security findings, the performance issues, the onboarding pain for the support rep — would have been smaller if that document had existed in month one. They were not smaller. They were not smaller because I did not write the document. I am sorry."

There was a silence. It was, I noticed, not an awkward silence. It was a *waiting* silence. The team was waiting for the next person.

The Tech Lead spoke next. He had not, when I started, been the next obvious speaker. He simply, at some point in my answer, decided that he would be.

"I did not review pull requests," he said. "For the entire project. I had a queue of, on average, between five and twelve open PRs at any given moment. I told myself I was 'reviewing them when I had time.' I did not have time. I never had time. I was always going to have time *next week*. The result was that — most notably — the protagonist," and here he gestured at me in a way I will remember for a long time, because in his own awkward and considered way he was using me as an example without using my name, "the protagonist self-merged the auth refactor, broke logins for seventy percent of our users, and worked through a night with the QA Tester to fix it. This was, in the final accounting, my fault. He should not have self-merged. But the only reason he had to was that I had, for two weeks, refused to look at his PR. I am sorry for that. To him, specifically. And to the QA Tester, who paid the cost in lost sleep that I, whose job it was to review the PR, did not pay."

The QA Tester was, I noticed, looking down at her notebook. She did not look up.

The next to speak was Developer 1.

"I committed the admin-true check," she said. Her voice was quieter than usual. "The one the security engineer found. I knew it was a placeholder. I told myself it was *temporary*. It was temporary in the sense that I was going to replace it. It was permanent in the sense that I never did. The security engineer is the only reason that line did not ship to production. I want to say, here, in this room, that I will never write a hardcoded auth check again. Not even as a placeholder. Not even *temporarily*. I have spent, since the security review, a lot of time thinking about how *temporary* is the most dangerous word in this profession. I would like to use it less."

Developer 3 went next. I had not expected him to speak at all. He had, throughout the project, been the quietest person on the team.

"I let the iOS provisioning certificate expire," he said. "Three times. Push notifications were dead for four days. The App Store review that said *did your company also die* was about that. I received the cert expiration warning emails. I read them. I marked them as read. I did not open them. I told myself I would deal with them on Monday. They were dated on Tuesdays."

He paused.

"I also," he said, "have not been doing my job. The mobile app exists, in the codebase, but it does not ship. I have, for six months, told the team it was *almost ready*. It is not almost ready. It is months of work. I should have said this in month two. I did not. I am saying it now."

The DevOps Engineer went next.

"I knew about the `exit 0` in the nightly script for two and a half years. I told nobody. I did not fix it because fixing it would have made the failures visible, and the failures being visible would have made the team angry at me, and I did not want the team to be angry at me, and so I did not make the failures visible, and the failures continued, invisibly, for two and a half years. This is, in the most direct possible way, my failure. I owe the team an apology. I owe the QA Tester, in particular, an apology, because she was working manually around a gap I had personally been hiding. I am sorry."

The QA Tester finally looked up. She did not say anything to the DevOps Engineer. She just nodded, once, the small, sad nod of forgiveness that takes years to grow back.

The Sales Manager was next. I will tell you, I had been afraid of his answer. The Sales Manager had been, on this project, the source of more pressure than perhaps any other person, and I had assumed, in my unkinder moments, that he would not have the self-awareness to answer this question honestly.

I was, again, wrong.

"I sold features that did not exist," he said. "I told clients that we were building things, in March, that nobody on the engineering team had agreed to build. I did not check before I told them. I told them because they wanted to hear it and I wanted to close the deal. The mobile app was one of those features. The nine-widget dashboard was another. The third-party integrations we still have not built were three more. When the engineering team came back, in month four, and said *we cannot build all of this*, I treated them as the obstacle. They were not the obstacle. The obstacle was me. I sold a product I did not, in any real sense, understand. I am, still, learning what we sold versus what we built. I owe the engineering team, specifically, an apology for treating their honesty as inconvenient. Their honesty was the only thing keeping me from selling, in month six, more things we could not build."

The Customer Support Rep was next. She had been, until that moment, watching the room with the specific careful attention of a person who has joined a family gathering as a new in-law.

"I will say only," she said, "that I have worked at this company for almost four months, and the first person from engineering to walk over to my desk was the protagonist, eighteen days before launch. I am not sure that is a question-four answer for me. I think that is a question-three answer for the rest of you. I do not say this with any anger. I say it because, if this room is for telling the truth, that is the truth I have."

There was a longer silence after this. Several people looked at the table.

"Thank you," I said, eventually. "That is fair. We will not let it happen again."

We went on. The Product Manager, on the video call, gave a slow and surprisingly honest answer about how he had used travel as a way of being *technically* involved while being

substantively absent, and how he was going to, on the next project, attend at least half of the kickoff meetings in person. The QA Tester went last. Her answer was the shortest of any of ours.

"I should have escalated harder," she said. "I should have, on at least three occasions, gone over the Tech Lead's head. I did not, because I was being polite. I will, on the next project, be less polite. I am sorry."

The room, after she finished, sat in a quiet that nobody seemed to want to break.

I looked around. I felt, in my chest, the specific small ache that is the only honest reward for running a meeting in which people have, finally, stopped pretending.

I read once, on a developer forum, the comment:

a real retrospective is the only meeting in software where the goal is for everyone to leave a little bit smaller than they came in. smaller in ego. smaller in defensiveness. smaller in the story they were telling themselves about who was to blame. when nobody leaves a meeting smaller, nothing has happened. the meeting was a costume. the work has not begun.

I closed the laptop. I closed the notebook. I said, gently:

"Thank you. All of you. We will do this every six weeks from now on. I will keep these notes. We will hold each other to the *one thing we will do differently*. We will not, please, ever again, ship a project for which the most accurate description is *I'll do it tomorrow*. We will not do that to ourselves. We will not do that to each other."

The Tech Lead said, very quietly, "Agreed."

We left the room at four-thirty. We had run, in the end, ninety minutes over. Nobody had left early.

It was the only meeting on this project nobody had left early.

It was, perhaps, the only meeting any of us had ever attended that was the meeting it claimed to be.

— END OF 6.2 —

6.3 — THE PROMOTION

"They asked me, in the promotion review, what was the most important thing I had built in the last year. I told them: a runbook. They laughed, politely. I did not laugh back. The runbook is, at this exact moment, being read by an on-call engineer at three a.m. in a city I have never been to. The runbook is doing, right now, what no piece of code I have ever shipped is doing. I think, on balance, the runbook wins."

— anonymous staff engineer, blog

The promotion happened, formally, six weeks after launch.

I say *formally* because, in the way these things really happen, the promotion had happened long before the paperwork. The promotion had happened on the day in month five when the QA Tester had said *okay* in response to my saying *I think I need to take over the project*. The promotion had happened in the eighteen days before launch when I had stopped writing code and started writing meeting agendas. The promotion had happened in the architecture document, in the runbooks, in the monitoring dashboard, in the apology to the Customer Support Rep, in the retrospective. The promotion was, by the time anyone offered it to me on paper, an acknowledgment of a shape I had already grown into.

The conversation happened on a Tuesday morning. The Director of Engineering — a woman I had spoken to, in the eight months I had been on this project, perhaps four times — pinged me at nine a.m. with a calendar invite for nine-thirty. The invite was titled, with the particular bureaucratic blandness of these things, *Career Discussion*.

I went to her office. She offered me coffee. She had, on her desk, a printed copy of the architecture document.

"This is good," she said.

"Thanks."

"This is unusually good. Most teams of your size do not have a document like this. The teams that have a document like this rarely have one this honest. The section called *Things That Will Bite Us Eventually* is, in my career, the most useful section I have ever read in any architecture document I have ever been handed. Including ones I wrote myself."

I did not know what to say to this. I said *thank you*.

"I have been talking to the Tech Lead," she said. "And the QA Tester. And, in a separate conversation, the DevOps Engineer. They have all, independently, said the same thing about the second half of this project, which is that the thing that turned the corner was not a piece of code. It was that you started running the project."

"I —"

"I want to formalize that. I would like to make you, formally, the project manager of this team, and the engineering lead of the next project. You will continue to write code, but, in the words of one of your colleagues, *much less, and much more carefully, and mostly only when nobody else can*. Your responsibility will be the team. Their growth. Their decisions. Their ability to ship. You will own the *what we ship and when* in partnership with the Product Manager. Your title will change. Your salary will change. The change to your salary will be in your direction. You will, I hope, be pleased."

I sat for a moment. I had known, on some level, this was coming. I had not, despite knowing, been able to imagine the shape of the conversation in advance.

"Can I ask one thing," I said.

"Yes."

"What happens to the Tech Lead."

She looked at me for a long second.

"He has been a Tech Lead for seven years," she said. "He is, by every metric I have, an excellent senior engineer. He has also, on every project he has led, been the bottleneck on code review. We have talked about this. He is going to step into a principal IC role on a different team. He has chosen this. He is, I think, relieved. He told me, two weeks ago, that he had been carrying a job he was not, in his deepest heart, suited for, and that watching you take it on had given him the courage to say so. Those were his words. Not mine."

I thought about the Tech Lead. I thought about his unread PR queue. I thought about how much of his silence, on this project, I had read as indifference, when in fact it had perhaps been the silence of a man who had been, for years, in the wrong job, and who had not known how to say so.

"I'm glad he chose that," I said.

"Me too."

She slid a piece of paper across the desk. I did not, at that moment, read it. I would read it later, in a quiet office, with the door closed, and I would stare at the title line for a long time, because the title line had a word in it that I had, in fifteen years, not expected to ever see attached to my own name.

I went back to my desk. I did not say anything to anyone. I sat down. I looked at the dashboard. The dashboard was green. The synthetic checks were passing. The error rate was at zero point three percent, which was, by our internal definition, healthy. The support queue had four open tickets, all triaged, all assigned, all under SLA.

The Customer Support Rep walked over. She was holding a small piece of paper.

"This was on my desk this morning," she said. "From the customer at pilot client #2. The one with the screen reader."

I took the piece of paper. It was a printed-out email. The subject line was *thank you*. The body said:

Hi — I just wanted to let you know that I logged into the new portal this morning. The login button was a button. The dashboard widgets had labels. The export modal did not trap my focus. I was able to do my job, today, in your software, without anyone else's help, for the first time in a year. I do not say this lightly. I have spent a long time having to ask people to read screens out to me. I did not have to ask anyone today. Please tell whoever did this that it mattered. To me. To the people on my team who use assistive tech. To the people I have not met yet who will, because of this, be able to use your product. Thank you.

I read the email three times.

I thought about the JIRA ticket I had let auto-close. The one that had said *backend ARIA support — propagate label metadata through the API*. The one I had marked *low priority*. The one I had said *I'll do it tomorrow* about, six times, until tomorrow had become *too late*.

I thought about how, in the second half of the project, I had finally written the API changes. I had written them on a Saturday afternoon, in week twenty-three, between the new monitoring rollout and the runbooks. I had written them quietly, without telling anyone, without putting them in any deck. I had simply opened the ticket, re-opened it, written the code, written the tests, gotten Developer 1 to wire up the frontend, and shipped them. The whole thing had taken, in the end, about eleven hours. *Eleven hours.* For something I had been deferring for six months.

That ratio is, I think now, the one I will think about for the rest of my career.

The Customer Support Rep was watching my face.

"Was that you," she said. "The accessibility work."

"Mostly. With Developer 1."

"I thought so."

I did not know what else to say. I handed her back the email.

"Will you forward this to the team," I said.

"Already done."

She smiled at me, briefly, and walked away.

We had cake that afternoon. The Customer Support Rep had organized it, without telling me, with the DevOps Engineer's help. The cake had no inscription. It was a small round cake from a normal bakery. We ate it at the long table by the window. The Tech Lead, who had, by then, accepted his new role on the other team, gave a short, awkward, self-deprecating speech that ended with him saying, in the most genuine voice I have ever heard from him:

"I am proud of all of you. I am proud, in particular, of the protagonist, who did the work that I should have been doing, and did it better than I would have."

He sat down quickly. He was embarrassed. He was, I think, also a little proud of himself for having said it.

The Sales Manager raised his coffee.

"To the protagonist," he said.

"To the protagonist," they said.

I felt, briefly, like crying. I did not. I did, however, look at the floor for slightly longer than the social norm allowed.

We had, the next morning, the first standup of the new project. It was a small, ordinary standup. We went around the table. We answered three questions:

— What did you do yesterday? — What are you doing today? — What will stop you from doing it?

When it got to me, I gave my own answer. Then I said, to the team, the thing I had been planning to say for several days.

"One more thing," I said. "I would like us to start a small ritual. It costs us nothing. It will feel, the first time, slightly silly. We are going to do it anyway."

I had their attention.

"At the end of every standup," I said, "we are going to go around the room one more time. Each of us is going to name one thing we have been *putting off*. Just one. It does not have to be big. It can be small. It can be embarrassing. The only rule is that it has to be true. And the only response we are allowed to give, to each person's answer, is the same single word."

"What word," the DevOps Engineer said.

"The word is *today*. As in: *I will do that, today*. Not *tomorrow*. Not *soon*. Not *this sprint*. Not *after the next milestone*. Today. The word *today* is the only word the room is allowed to say back. If you cannot, in good conscience, say *today*, then you do not get to name a thing in the ritual. You sit out the day. You think about it. You come back tomorrow, and you name something you can actually, truly, do today."

There was a small silence.

"That sounds," Developer 1 said, "really annoying."

"Yes."

"We are going to do it anyway."

"Yes."

She looked at me. She nodded.

"Okay," she said. "I'll go first. The thing I have been putting off is that I have a comment on my own PR from yesterday that I have not addressed. I will fix it."

"Today," the room said.

It was, I will tell you, the first time in my career that I had heard an entire team of engineers say a single word in unison and not feel, immediately, that the word was a lie.

The DevOps Engineer went next. He named a runbook section he had been meaning to update. The room said *today*. The Customer Support Rep — who was in our standup now, every morning, by my standing invitation — named a macro she had been meaning to write for a recurring class of ticket. The room said *today*. Developer 3 named the email he was going to send about scoping the mobile app properly. The room said *today*. The QA Tester, last, named a flaky test she had been meaning, for two weeks, to either fix or delete.

"Today," we said.

We went back to our desks.

It is, as I write this, sixteen weeks later. We have held the ritual every morning. We have, between us, named seven hundred and forty things, by my rough count. About six hundred of them have, in fact, been done that day. The other hundred and forty, when not done, have been honestly named the next morning as the same item, again, until they were either done or admitted as out-of-scope and removed from the list.

The team, since the launch, has shipped four major features. Each of them on time. Each of them with real tests. Each of them documented. Each of them with runbooks. Each of them launched on Tuesdays.

The promotion paperwork, on my desk, is now a little faded at the corners. I keep it there. I look at it, sometimes, when I am tired, and I think about the person I was on the first day of this project. The one who, on a Monday morning in the *Innovation Suite*, listened to the UX Designer talk about Frustrated Fiona, and who heard, somewhere in the back of his head, the small voice that said *I should write this down*, and who, instead, said nothing, and who let the moment pass, and who promised himself he would do it tomorrow.

That person is still, in some part of me, here. He will always, in some part of me, be here. He is not, any more, the loudest voice. There is a louder one now. The louder one belongs to the team. The louder one, every morning, says one word in unison.

The word is *today*.

It has, against everything I would have predicted at the start of this story, turned out to be enough.

— END OF CHAPTER 6 —

EPILOGUE — TWO YEARS LATER

> "Years after the project ended, I found, in an old notebook, a page with the words 'I'll do it tomorrow' written forty-seven times in my own handwriting. I do not remember writing it. I remember every single thing I was avoiding when I did. I keep the notebook on my desk. It is, on most days, the most useful piece of management literature I own."
>
> — anonymous staff engineer, blog

It is, as I write this, two years after the events I have described.

The Customer Portal still runs. It has, in those two years, shipped fourteen major features. It has, in those two years, had three customer-impacting outages, none of which lasted more than nineteen minutes, and all of which were detected by our monitoring before any customer reported them. It has, in those two years, grown from three thousand customers on launch day to, as of last quarter, just under forty thousand. The mobile app, which Developer 3 had honestly admitted, in the retrospective, was months from real, took, in fact, seven months. It shipped on a Tuesday. The launch was, like all of our launches now, the most boring kind. The runbooks were ready. The dashboards were green. The Customer Support Rep was in the standup the morning of, with a sheet of pre-written ticket replies for the three classes of question we expected. We got, in the first week, two of those three classes. The third never came. We had over- prepared. The Sales Manager, when I told him, said, "I have come to like over-preparing." It was the only unprompted compliment I have ever heard him pay an engineering process.

I am still the project manager of the team, in a formal sense. I am, in a less formal sense, also still a backend developer. I write, on average, perhaps two pull requests a week. They are small. They are surgical. They are usually in the parts of the codebase nobody else has the context for, which I have been steadily transferring, by way of documentation and pairing sessions, to Developer 1 and to a new backend hire we made nine months ago, who is, I am quietly pleased to report, much better at the streaming consumer than I ever was. The wrapper around the legacy framework — the war crime against software engineering, which I am still slightly proud of, though I would not put it on a resume — has, in the second year, finally been retired. We replaced it with a real streaming framework. It took a sprint. It would have taken, had we done it on the original project, four sprints. The difference is the documentation. We knew, by the time we did the migration, exactly what the wrapper had been doing and why. The *Things That Will Bite Us Eventually* section of the architecture doc had, on the day we replaced the wrapper, been quietly editable for two years, and the line about it had had, by then, three different owners, and each owner had updated the entry, and

the entry had been read, and the read had become a plan, and the plan had become a PR, and the PR had been reviewed within forty-eight hours, because that is now how we do things.

The team has changed. The Tech Lead, now a principal IC on a different team, sends me, every few months, a small note about something he has read or watched or thought about that he believes might be useful. He has, in the last year, sent me a podcast, two papers, and a screenshot of a single Slack message somebody had sent in his current team that had reminded him, he said, of the architecture doc. He has not, as far as I can tell, missed leading people. He has, as far as I can tell, found his real shape. We have lunch, when our calendars permit it, about once a quarter. The lunches are short. They are, in their quiet way, the most professionally useful conversations I have.

The Product Manager left the company eleven months after launch. He took a role at a smaller company, in a city he had been wanting to live in. He sent the team a short goodbye email. The email was, by the standards of his prior communication, unusually present. It said, in part:

I was not, on this team, the manager you needed. I understand this now. I am taking that lesson with me. The next team will get a better version of me because of you. Thank you for being patient with the worse one.

The team replied with, collectively, nineteen heart emojis and one mildly skeptical thumbs-up from the Sales Manager. I think the thumbs-up was a kindness. I think the Sales Manager had, in his own way, learned to be kind by then.

The Sales Manager is, against all odds, still in the same role. He is, by every measure I can name, a different human being than he was on the kickoff Monday. He attends the daily standup. He reads the support tickets. He has, by my count, twice in the last year personally walked a prospective client through a feature we had *not yet shipped*, and explained, in plain language, that we would have it in six weeks, and then, six weeks later, asked us in the standup if we needed anything from him to make the date. He is, on his quieter days, the closest thing the team has to a customer advocate, which is a sentence I would not have, on month four of the original project, believed I would ever write. People can change. I would not have believed that either.

The Customer Support Rep is now leading a small team of three. She runs the daily 9:15. The first item on the 9:15 agenda, every day, is the question *what did the customers tell us yesterday that we did not already know?* The agenda has, in the eighteen months it has been running, surfaced the root cause of, by my rough count, the four most useful product decisions we have made. None of those decisions came from a deck. All of them came from a paying customer who said something specific about something specific, and were heard by a person who had been listening, and were carried by that person into a room where engineers were also listening. This is, on some days, the entire mechanism by which our product gets better. I do not know how I ever thought it could be otherwise.

The QA Tester is, last year, no longer a QA tester. She is the Director of Engineering Quality, which is a title the company invented for her after she had, for the third quarter in a row, prevented a launch from being ruined by something an entire team of senior engineers had agreed they were "fine" with. The new title is, by her own description, "the same job with a slightly better email signature." She still keeps the notebook she had on the original project. She showed it to me, once, in a one-on-one. I will not tell you what is in it. I will tell you that, on the inside cover, she has written, in her small careful handwriting, the words *common, not normal*. I had not, until that moment, realized that I had taken those three words from her. I told her so. She said, dryly, "I know."

Developer 1 is still on the team. She has not, in two years, shipped a single placeholder auth check. She has, in those two years, refactored three different parts of the codebase that the rest of us were quietly afraid to touch. She has not asked permission. She has shipped each of them with a description and a test suite and a runbook update. She is, in the last year, also a tech lead on a small adjacent project. I gave her the role. The Tech Lead, when I told him, said, "Good. I should have done that two years ago." I did not say, *yes, you should have*. I did not need to. He already knew.

Developer 3 still owns mobile. Mobile is now a real team of three. The provisioning certificates are now on a corporate Apple ID, with two backup contacts, and there is, in the team channel, a bot that posts, every sixty days, the words *certificates expire in thirty days, who is renewing them?*. Developer 3 was the one who set up the bot. He set it up the week after the retrospective. I watched him do it. He did not say anything to anyone about doing it. He just did it. He named the bot, with what is, I have come to believe, the only joke he has ever made on company time, *OvermorrowReminder*. The bot still runs. The bot has not, since it was set up, ever fired in vain. The bot has also, as far as I can tell, become beloved.

The DevOps Engineer is, this quarter, on a sabbatical. He had not, in seven years at the company, taken one. We made him take it. He resisted. We told him, in the specific language he had taught us, that *the runbook exists*. He went. He sent us, three days into the sabbatical, a single message in the team channel, which said only:

DEVOPS: i hate that the runbook works. but i love that the runbook works. these feelings are not contradictory. don't page me.

We have not paged him. We will not page him. The runbook does, in fact, work.

The standup ritual is still in place. We have run it, every business day, for two years. It has spread, by quiet osmosis, to four other teams in the company. The Director of Engineering, who in that first conversation had handed me the promotion paperwork, has, on at least two occasions I know of, demonstrated the ritual to visiting executives from other companies. She has, on at least one of those occasions, called it, slightly exaggerating, *our cultural innovation*. It is not a cultural innovation. It is, as I told her in private afterward, a small daily reminder, with a one-word response, that the team has agreed, again, in front of each other, to be the team it intends to be. *Today.* That is all it is. The fact that it works does not,

in my opinion, make it an innovation. It makes it a habit of honesty. The two are, in software, often confused.

I have, in the two years since the original project, caught myself many times about to say *tomorrow*. I am not, despite everything, cured. The cure is not a thing this profession offers. The cure is a thing this profession arranges, daily, in small social rituals that do not let you slip back into the comfortable lie. The ritual works for me because it is *public*. The ritual works for me because, at nine-fifteen in the morning, in front of seven other people, I would rather name a small embarrassing thing I have been putting off than let those seven people see me pretend that I have been putting nothing off. Pride, in this case, is being made to do the work of virtue. I will take it. I have, after fifteen years of trying to use virtue alone, no further opinion about whether it ought to be virtue or pride that does the work, as long as the work, on the day, gets done.

I read once, on a developer forum, in a thread titled *what is the most important thing you have learned in your career*, the comment:

the senior engineer is not the one who writes the most code. the senior engineer is not the one who knows the most about the system. the senior engineer is the one who, when an unspoken thing is in the room, says it out loud first. everyone else is, by definition, junior, regardless of their title. the seniority is the saying. it is not the years.

I had, on the day I first read that comment, almost exactly fifteen years of experience. I had, on the day I first read that comment, never once been the person who said the unspoken thing first. I read the comment. I closed the tab. I told myself I would think about it tomorrow.

I did not, of course. I did not think about it for two more years. I thought about it, eventually, in front of an accessibility auditor and a list of two hundred and forty-one issues, with the QA Tester eating crackers in the next chair, on a Tuesday afternoon that felt, somehow, like the only Tuesday afternoon that had ever happened to me.

It is still happening to me, in some quiet way, every day. It will, I suspect, keep happening for as long as I do this work.

I am, in the morning, going to the standup.

Somebody will ask me what I have been putting off.

I will name something.

The team will reply with one word.

The word is *today*.

It is, on most days, still enough.

— END OF EPILOGUE —

— END —