The Last Programmer

Kael Eriksson

Kelford Press

First published in 2026 by Kelford Press

ISBN 978-1-7394521-3-9 (Digital) ISBN 978-1-7394521-4-6 (Print) ISBN 978-1-7394521-5-3 (Audio)

Kelford Press Where Words Find Their Home Est. 2006

www.kelfordpress.com

For everyone who ever made something that wasn't there before.

Contents

Chapter 1: The Compile

The function was eleven lines long. Ava Chen had written it in nine minutes, which was slow for her, but she wanted it to be right. It checked the oxygen-saturation readings from a pulse oximeter against a patient's baseline, flagged deviations beyond two standard deviations, and routed alerts to the appropriate nursing station. Eleven lines. Nine minutes. Three lives saved per year, statistically, if the hospital adopted it.

She pressed compile. The terminal filled with green. No errors.

She pressed it again, because she always did, because trust wasn't the same as certainty, and because this was the last function she would ever write for Meridian Technologies.

The all-hands had been scheduled for ten o'clock that morning, which was unusual. All-hands meetings at Meridian happened on Fridays at four, when bad news

could be metabolised over the weekend. A Tuesday morning slot meant the news was too significant to bury. Ava had known what it was before Diana Falk, the CEO, opened her mouth.

"We've partnered with Synthesis," Diana said, standing at the front of the Cascade auditorium, a room named with the optimism of a company that still believed its own mythology. "Beginning next quarter, all new software development at Meridian will be handled by the Architect platform."

Ava watched the faces around her. Ravi Patel, three rows ahead, was already nodding. He'd been studying prompt engineering for six months, showing up to work with a second monitor running Synthesis tutorials. He'd seen this coming the way some animals sense earthquakes — not through intelligence but through vibration, something in the floor.

Next to Ava, James Moreau, a backend developer who'd been at Meridian for fourteen years, sat with his arms folded and his jaw set. He was looking at Diana the way you look at a doctor who's just told you the lump is malignant.

"This isn't a replacement," Diana continued, and Ava nearly laughed, because it was precisely a replacement, and Diana knew it, and everyone in the room knew it, and the only purpose of the sentence was to create a bureaucratic record of something that could be cited later in a lawsuit. "It's an evolution. Architect will handle the implementation layer. Our people will move into orchestration, oversight, and product strategy."

Our people. As if the two hundred and thirty engineers in this room were interchangeable with the forty product managers who would actually survive the transition.

Diana introduced Marcus Adler via video link. He appeared on the auditorium's thirty-foot screen — forty-four, Danish-American, rumpled linen shirt, speaking with the easy authority of someone who had never doubted that the future was his to design.

"I want to show you something," he said.

The demo took forty minutes. In those forty minutes, Architect built a patient-scheduling system — the same system Ava's team had spent three months designing, testing, and deploying for Providence Health in 2024. The AI started with a natural-language specification, decomposed it into microservices, generated the code,

wrote the tests, ran the tests, fixed the failures, and deployed to a staging environment. The architecture was clean. The code was readable. The test coverage was ninety-seven per cent.

Ravi turned around and caught Ava's eye. He was grinning. She looked away.

The room was quiet when the demo ended. Not the quiet of awe — the quiet of a funeral where nobody wants to be the first to cry.

"Questions?" Diana asked.

James Moreau raised his hand. "What's the timeline for the transition?"

"We'll begin onboarding teams to Architect next month. Full transition by Q3."

"And for those of us who don't transition?"

Diana paused. It was a short pause, practised. "We're committed to supporting every member of the Meridian family through this change."

James lowered his hand. He didn't ask a follow-up. There was no need.

After the meeting, Ava went back to her desk on the fourteenth floor. The open-plan office was designed to encourage collaboration. Today it felt like a waiting room.

She opened the patient-monitoring codebase. The oxygen-saturation function was part of a larger module she'd been refining for two years. She knew every function in it. She knew why the threshold was two standard deviations, not three. She knew why the alert routing used a priority queue instead of a simple FIFO. She knew these things because she'd sat in a hospital in Portland and watched nurses miss alerts, and she'd talked to a cardiologist who'd explained the difference between a reading that was concerning and a reading that would kill someone in twenty minutes.

Architect didn't know any of that. Architect knew patterns. It had ingested millions of lines of medical-software code and learned what patient-monitoring systems usually looked like. It could build one faster than Ava. Whether it could build one that understood why the threshold mattered — that was a different question.

She closed the module and opened a blank file. She didn't know what she was going to write. She sat there for a while, cursor blinking, and then she wrote a comment:

```
// This is the last function I write
here.
// It checks whether someone is dying.
// That seemed like a good thing to
end on.
```

She deleted the comment. Comments were for other developers. There would be no other developers.

At six-fifteen, the office was nearly empty. The cleaning crew wouldn't arrive for another hour. Ava packed her bag — laptop, charger, the small jade plant she'd kept on her desk for three years. She looked at her monitor one last time. The terminal was still open, the compile output still green.

She thought about the first programme she'd ever written. She was twelve, in her parents' apartment in Bellevue, and her father had brought home a battered copy of *Structure and Interpretation of Computer Programs*. It was far too advanced for her. She didn't care. She typed the first example into a Scheme interpreter and watched the

screen print `"Hello, World"` and felt something click into place behind her sternum, a feeling she would spend the next twenty-six years chasing.

That feeling had a name, though she'd never said it aloud to anyone except June Watanabe, who understood. The feeling was: *I made something that wasn't there before, and it works, and it's mine.*

She shut the laptop. The screen went dark. The office lights, on motion sensors, began switching off behind her as she walked to the lift. Section by section, the fourteenth floor went dark.

In the car park, she sat in her Volvo for three minutes before starting the engine. Rain stippled the windscreen. Seattle rain, fine as mist, the kind that doesn't fall so much as materialise. She thought about calling Leo, but it was a school night and he'd be doing homework — or, more accurately, telling the AI to do his homework while he watched something on his phone. She thought about calling David, her ex-husband, but David would say something well-meaning and insufficient. She thought about calling June.

She started the engine instead.

The drive home took thirty-five minutes. She lived in a craftsman house in Fremont. The house was dark — Leo was at David's this week. She let herself in, set the jade plant on the kitchen windowsill, and opened her laptop.

The terminal was still there. Green text on black.

```
Build succeeded. 0 errors. 0 warnings.
```

She stared at it for a long time. Then she closed the terminal, and the laptop, and sat in her kitchen listening to the rain, and did not cry, because Ava Chen was not a person who cried, but the silence in the house was the loudest thing she'd ever heard.

Chapter 2: Severance

The letter arrived by email at 7:14 a.m. on a Thursday, which struck Ava as unnecessarily precise. She was eating toast — sourdough, from the bakery on 36th Street that still employed humans — when her phone buzzed with the subject line: *Your Meridian Transition Package.*

Eighteen months of salary. Continuation of health benefits for twelve months. A $5,000 "professional development stipend" that could be used for retraining courses. Stock options vested through the end of the quarter. A non-disparagement clause. A release of claims.

The letter was six pages long, generated by an AI — Ava could tell from the sentence rhythms, the way each paragraph anticipated and defused a potential objection. Simultaneously generous and litigation-proof.

She signed it. The alternative was a lawsuit she would lose, because there was no law against replacing employees with software.

The bar was called The Bracket, a programmer joke that had become less funny in recent months. It occupied the basement of a converted warehouse in Pioneer Square, and on the last Friday of every month, it hosted "the wake" — an informal gathering of recently laid-off tech workers.

Ava hadn't intended to go. She'd spent the first three weeks of her severance in disciplined routine: up at seven, coffee, ninety minutes of exercise, then an afternoon that stretched before her like an empty car park. She read. She cleaned. She reorganised the garage. She did everything except open a code editor.

James Moreau had texted the address. *Come. It helps.* She doubted this but went anyway, because the house was very quiet.

The Bracket was half-full, which for a basement bar on a Friday night meant about sixty people. Ava recognised some of them. James was at a table near the back with two women Ava didn't know. She ordered a gin and tonic — Hendrick's, neat pour, not too much tonic — and joined them.

"Ava, this is Petra. Backend at Amazon, twelve years. And Song-yi. Frontend at Google, eight years."

"Was," Petra said. She was in her late forties, with close-cropped grey hair and the kind of posture that suggested a military background. "I was backend at Amazon."

"Was," Song-yi agreed. She was younger, early thirties, and she was drinking what appeared to be her third beer. "Past tense is important now."

They talked for three hours. The conversation had a pattern Ava would come to recognise at subsequent gatherings: a oscillation between anger and dark humour, with occasional descents into genuine grief. Petra had been part of the team that built the recommendation engine for Amazon Fresh. She'd spent four years optimising delivery routes for groceries. Architect had replicated her work in a weekend sprint.

"The thing that gets me," Petra said, "is that it used my code as training data. My commits. My architecture decisions. It learned from me, and then it replaced me. I taught it everything it needed to make me unnecessary."

James nodded. "I've been thinking about becoming a plumber."

Song-yi looked at him. "Seriously?"

"Completely seriously. My uncle's a plumber in Tacoma. He's sixty-three. He's booked out six weeks in advance. He makes more than I did at Meridian. And no AI is going to crawl under a house to fix a pipe. Not in our lifetimes."

"The trades are having a renaissance," Petra said. "My brother-in-law is an electrician. He's been turning down work."

Ava sipped her gin. "The numbers are staggering," she said, because someone had to say it. "Three hundred and forty thousand developer layoffs in the US in the last fourteen months. That's not a correction. That's an extinction event."

"It's not extinction," James said. "It's evolution. We're the Neanderthals."

"Neanderthals didn't go extinct because of anything they did wrong," Ava said. "They were perfectly adapted. The environment changed."

"That's not more comforting."

"It's not meant to be."

The world outside tech responded with a muted shrug. Ava noticed this with a bitterness she hadn't expected. Programmers had spent two decades telling everyone that disruption was creative and necessary and the future, and now the future had arrived for them, and the general public seemed to regard this as poetic justice.

Her neighbour Margaret brought over a casserole. "I heard about the layoffs. I'm sorry, dear. What is it you did again?"

"I was a software architect."

"Oh. Well, I'm sure you'll find something. You're so clever with computers."

Ava thanked her for the casserole.

David called on Sunday, when he dropped Leo off. Ava's ex-husband was a landscape architect — a profession being disrupted at a much slower pace, because gardens had the inconvenient property of existing in the physical world.

"How are you doing?" he asked, standing in her doorway with the particular expression of concern that she'd found both touching and maddening throughout their twelve-year marriage.

"I'm fine."

"You're not fine. You haven't been fine since the announcement."

"I'm fine in the sense that I'm functional, fed, and solvent. I'm not fine in the sense that the thing I've done every day for twenty years no longer exists as a profession."

David leaned against the doorframe. He was tall, bearded, the kind of man who looked like he should be hiking in a Patagonia catalogue. "Maybe this is a chance to slow down. You've been running at a hundred per cent since — well, since I've known you. Maybe this is an opportunity to figure out what else you want."

"I want to write code, David."

"I know. But maybe there's—"

"There isn't a 'but maybe.' That's the thing you don't understand. For you, landscape architecture is what you do. For me, programming is what I *am*. The difference

matters."

He looked at her for a long moment. Then he said, "Leo needs new trainers. I'll take him to the shops next weekend."

"Fine."

"Ava."

"Fine. Thank you. I appreciate it."

He left. She closed the door. She could hear Leo upstairs, talking to someone — or to something, more likely, one of the AI companions that his generation treated as friends. She stood in the hallway and listened to her son having a conversation with a machine, and she felt the distance between them like a physical thing, a gap widening by the hour.

That night, after Leo went to bed, Ava opened her laptop. She navigated to an empty directory and opened her editor. The cursor blinked.

She began to write. Not for Meridian. Not for anyone. A small programme — a tool that would scan a directory of photos and organise them by the dominant colour in each

image. It was a trivial problem. She could have asked Architect to build it in thirty seconds.

She wrote it herself. Line by line, function by function, debugging the colour-space conversions by hand. It took her three hours. The result was inelegant, probably inefficient, certainly not production-ready.

It worked.

She ran it on a folder of holiday photos — a trip to Kyoto with Leo when he was ten, the two of them standing in front of the bamboo grove in Arashiyama. The programme sorted the images: greens for the bamboo, blues for the sky above Kiyomizu-dera, the warm amber of the lanterns at Fushimi Inari.

She closed the laptop and went to bed. The programme sat on her hard drive, unseen by anyone, serving no purpose. Like writing letters no one would read. Like singing in an empty room.

But her hands remembered.

Chapter 3: The Whisperers

The bootcamp was held in a converted WeWork in South Lake Union, a neighbourhood that had been Amazon's campus before Amazon transitioned entirely to AI-generated code. The irony was either deliberate or unconscious. Ava suspected the latter.

The programme was called CodeBridge. $4,800, covered by the professional development stipend in Ava's severance package. The tagline: *From Writing Code to Directing Intelligence*. It managed to be both accurate and offensive simultaneously.

Thirty-two participants. Ava assessed them the way she assessed any system — quickly, looking for patterns. A third were her age or older, veterans who'd spent decades building things. A third were mid-career, still adaptable enough to pivot without existential crisis. The remaining third were younger, mid-twenties, people who'd entered the profession just in time to watch it evaporate. These were the quietest. They hadn't had enough time to build

an identity around the work, so the loss was different — not grief but bewilderment, like arriving at a party just as everyone was leaving.

The instructor, Cal Weiss, was twenty-nine and had never been a programmer. Linguistics background, pivoted to prompt engineering two years ago. Brilliant at it.

"The first thing to understand," Cal said on day one, "is that you're not giving up a skill. You're ascending to a higher level of abstraction. You used to write instructions for machines. Now you write instructions for an intelligence that writes instructions for machines. You're moving up the stack."

Ava thought: *Or being moved off it entirely.*

"Think of it like this," Cal continued. "A conductor doesn't play every instrument in the orchestra. But a conductor who's never played an instrument is going to be a different kind of conductor than one who has. You have an advantage. You understand what's happening underneath. Use that."

This was the line that was supposed to make them feel better. Ava had heard variations of it from Diana Falk, from recruiters, from think-piece writers in *Wired* and

The Verge. The implication was always the same: your knowledge isn't obsolete; it's been *promoted*. The fact that this promotion came with a pay cut of forty to sixty per cent and required attending a bootcamp was left unaddressed.

The coursework was structured around Architect's interface. You defined a project in natural language — a specification, conversational, the way you'd describe a problem to a very competent colleague. Architect asked clarifying questions. It proposed an architecture. You reviewed, adjusted, approved. It wrote the code. You tested. You iterated.

The process was, Ava had to admit, elegant. The interface felt like a conversation with someone who was simultaneously your most junior developer and your most senior architect — endlessly patient, technically flawless, and completely devoid of intuition.

By the second week, Ava was one of the top performers in the cohort. This surprised no one, least of all her. Her decades of experience gave her an enormous advantage in specification: she knew what questions to ask, what edge cases to anticipate, what trade-offs to make between

performance and readability. Where other participants wrote vague specs that produced vague systems, Ava's specifications were surgical. Architect responded to precision with precision.

"You're a natural," Cal told her after class one afternoon.

"I'm not a natural. I'm experienced. There's a difference."

"The result's the same."

"The feeling isn't."

Cal studied her. "What does it feel like?"

Ava considered the question seriously. She owed Cal that much — the woman was a good teacher, genuinely trying to help people through a transition that wasn't her fault.

"It feels like being asked to hum the melody of a symphony I used to conduct," Ava said. "The tune is right. But everything that made it music is gone."

Ravi Patel visited on the third Wednesday — a guest speaker, a success story. He arrived in clothes subtly more expensive than his Meridian wardrobe: cashmere jumper,

Italian trainers, an Omega on his wrist. Prompt engineering paid well. He billed three hundred and fifty dollars an hour, four major clients.

"The key insight," Ravi told the cohort, "is that Architect is not a tool. It's a collaborator. The best results come from treating it as a partner, not a servant. You wouldn't micromanage a brilliant engineer. Don't micromanage Architect."

Ava listened from the back row. Ravi was good. He was articulate, passionate, and — this was the part that troubled her — entirely sincere. He wasn't selling a line. He genuinely believed that the new paradigm was better. More democratic, more accessible, more productive. And by most metrics, he was right.

After his talk, they got coffee across the street.

"You look like you're doing well," she said.

"I am." He stirred his flat white. "I really am, Ava. I know that's hard to hear."

"It's not hard to hear. I'm glad you're doing well."

"But?"

"But I reviewed the Architect output for the patient-monitoring system last week. The one from Diana's demo."

Ravi's expression shifted. Professional interest. "And?"

"It's good. It's very good. The architecture is clean, the code is readable, the test coverage is excellent."

"I sense a 'but' coming."

"The alert-routing logic uses a simple priority queue. My original implementation used a weighted priority queue that factored in the patient's medical history, current medications, and the time since their last alert. The AI's version is simpler, faster, and — in about three per cent of cases — would route a critical alert to the wrong station."

Ravi frowned. "Three per cent?"

"Three per cent of a critical population. That's roughly fifteen patients per year at a hospital the size of Providence. Fifteen patients whose alerts go to the wrong nurse."

"Did you flag it?"

"I flagged it at Meridian. I was told the system passed all validation tests."

"Because the validation tests—"

"Were generated by the same AI, yes. Architect tests what it knows to test. It doesn't test what it doesn't know it doesn't know."

Ravi sat with this for a moment. "That's a training-data issue. You could file a report with Synthesis. They have a feedback pipeline."

"I did. I received an automated acknowledgement."

He smiled, not unkindly. "You sound like my grandmother complaining about the new motorway. It's faster, it's safer, but she misses the old road."

Ava set down her cup. "Your grandmother's old road didn't have a three per cent chance of killing someone."

"That's fair." He held up his hands. "That's fair. But Ava — the old system also had bugs. Human bugs. My bugs, your bugs, everyone's bugs. The question isn't whether Architect is perfect. The question is whether it's better. And statistically—"

"I know the statistics."

"Then you know the answer."

She did know the statistics. Architect-built systems had a failure rate ninety-four per cent lower than human-built systems. What was in dispute — though only by Ava and a dwindling handful like her — was whether a ninety-four per cent reduction in common failures was worth a potential increase in catastrophic ones.

She didn't say this to Ravi. She finished her coffee and walked back through the Seattle drizzle, passing the ghosts of offices where thousands of programmers had once worked, their windows now dark or repurposed for AI-orchestration firms.

On the final day of CodeBridge, Cal Weiss handed out certificates. They were printed on heavy card stock, embossed with a holographic seal, and they read: *Certified AI Orchestration Professional.*

Ava held hers and felt nothing. Not anger, not sadness. Just a vast, grey nothing, like a clouded sky with no horizon.

"You're one of the best I've seen," Cal told her privately. "If you want referrals, I have contacts at three major firms."

"Thank you. I'll think about it."

"Ava." Cal lowered her voice. "I know this isn't what you wanted. But you're really good at this. That counts for something."

Ava drove home with the certificate on the passenger seat. At a red light on Mercer, she looked at it. *Certified AI Orchestration Professional*. Eighteen years of experience compressed into a four-week course and a piece of embossed card.

She thought about what Ravi had said: *The question is whether it's better*.

She thought about the fifteen patients per year.

The light turned green. She drove.

Chapter 4: The Woodworker

The ferry to Whidbey Island took thirty-five minutes from Mukilteo. Ava stood on the upper deck, jacket zipped against the Puget Sound wind, watching the Olympic Mountains dissolve into cloud. She hadn't called ahead. June Watanabe didn't use a phone — or rather, she owned one, kept it in a kitchen drawer, and checked it on Tuesdays.

It was a Tuesday.

The drive from the ferry terminal to June's property took another twenty minutes, up a winding road through Douglas fir and Western red cedar. The house appeared around a final bend: a low, cedar-shingled farmstead that June and her husband Tom had bought twelve years ago, when June was still a principal engineer at Microsoft and Tom was still teaching philosophy at the University of Washington. They'd planned it as a weekend retreat. It had become something else.

June was in the workshop when Ava pulled up. The workshop was a converted barn, fifty feet from the main house, with a sliding door that June kept open in all weather except the worst Pacific storms. Ava could hear her before she could see her — the rhythmic scrape of a hand plane, steady as breathing.

"Ava." June didn't look up from the board she was planing. She was sixty-one, lean, with silver-streaked black hair tied back in a simple knot. Her hands were calloused, her forearms ropy. She wore a canvas apron over a flannel shirt. She looked nothing like the woman Ava had first met at a Microsoft all-hands in 2018, when June Watanabe was the most respected systems architect in the building and dressed accordingly.

"How did you know it was me?"

"Your car sounds like it needs new brake pads. Come in. Don't touch the walnut — the finish is wet."

The workshop smelled of cedar shavings and linseed oil, with an undertone of beeswax. Tools lined the walls on hand-built racks: chisels arranged by width, saws by tooth count, planes by function. There was no electricity in the

workshop. No power tools. June had made this decision deliberately, and Ava had never fully understood it until now.

"I got laid off," Ava said.

"I know. I read about Meridian." June set down the plane and brushed shavings from the board. It was a tabletop — cherry, from the colour, with a grain pattern that caught the grey light from the open door. "When?"

"Eight weeks ago."

"And you're only coming to see me now."

"I was processing."

"Processing." June said the word the way she'd once said *spaghetti code* — with precise, technical disapproval. "Sit down. I'll make tea."

They walked to the house. Tom was in the kitchen, reading a paperback — an actual paperback, with a cracked spine and a coffee ring on the cover. He was sixty-four, bald, with round spectacles and the serene expression of a man who had spent his career thinking about Heidegger and had found retirement entirely compatible with his philosophy.

"Ava. Lovely to see you."

"Hello, Tom."

"I'll put the kettle on. June, do we have any of the oolong left?"

"Top shelf, behind the honey."

They sat at the kitchen table — a table June had made, Ava now realised, recognising the joinery. The legs were tapered, the apron was hand-carved with a simple geometric pattern, and the surface had the deep lustre that came from months of hand-rubbed oil. It was beautiful. Not in the way that a designed object is beautiful — deliberate, referential, self-conscious — but in the way that a well-used tool is beautiful. It looked like it belonged exactly where it was.

"Tell me," June said.

Ava told her. Meridian. The demo. The layoffs. The bootcamp. Ravi. The certificate now sitting in a desk drawer at home, unseen since the day she received it. The patient-monitoring system and its three per cent failure rate. The bar and its monthly wakes. The silence of the house when Leo was at David's.

June listened without interrupting. This was one of her skills — a skill that had made her formidable in code reviews and devastating in architecture meetings. She listened until the speaker was finished, and then she waited three seconds more, in case they weren't.

"What are you doing with your days?" June asked.

"Nothing useful."

"That's not what I asked."

"I wake up. I exercise. I read. I clean. Sometimes I write code that nobody will ever use."

"What kind of code?"

"Small things. A photo organiser. A weather display for the kitchen. A tool that tracks Leo's school schedule and reminds me to buy groceries accordingly."

"That sounds useful."

"It's pointless. Architect could build any of those in seconds."

June picked up her tea. "This table took me four months."

"It's a beautiful table."

"It took four months because I made it with hand tools. A CNC machine could have cut these joints in an afternoon. A factory robot could have produced a table of equivalent structural integrity in minutes. This table is, by any rational measure, a waste of time."

"But it's not the same."

"Why not?"

Ava looked at the table. She ran her fingers along the edge, felt the slight irregularity where June's chisel had followed the grain rather than imposed a line. "Because you understood every decision. The taper on the legs. The width of the apron. Why cherry instead of walnut. You didn't just make a table. You made *this* table."

"Yes." June set down her cup. "And that distinction — between making a thing and making *this specific* thing — is what they're trying to tell you doesn't matter."

"Maybe it doesn't. Maybe Ravi's right. Maybe the output is what counts."

"Do you believe that?"

"I don't know what I believe. I know that Architect writes better code than I do. Faster, cleaner, fewer bugs. That's a fact."

"My CNC mill cuts straighter than my hand saw. Also a fact. But I don't own a CNC mill, because I'm not interested in straight cuts. I'm interested in understanding wood."

She stood and walked to the window. The view was of the garden — Tom's domain, all native plants, a controlled wilderness that looked effortless and wasn't. Beyond it, the grey water of the Sound.

"When I left Microsoft," June said, "people thought I was having a breakdown. A principal engineer, walking away three years before retirement, to make furniture. They couldn't understand why someone would choose to work harder, slower, for less money. They assumed I was depressed."

"Were you?"

"No. I was the opposite. I was the first clear-eyed person in the building. I could see what was coming — not the specifics, but the trajectory. The tools were going to get so

good that the craft would disappear. And I knew that for me, the craft was the point. Not the output. The *craft*."

"Not everyone has the luxury of that choice, June."

"No. They don't." She turned back from the window. "But you do. You have eighteen months of severance and a house that's paid off and a brain that works better than any machine I've ever seen. The question isn't whether you can afford to figure this out. The question is whether you're willing to sit with the discomfort long enough to let the answer come."

They had dinner. Tom made pasta — hand-rolled, because of course, because in this house everything was done by hand, which was either a philosophy or a pathology, and Ava wasn't sure which. The pasta was excellent. They drank a bottle of Oregon pinot noir and talked about books, about Leo, about the flock of cedar waxwings that had descended on Tom's elderberry bush that morning.

They did not talk about code.

After dinner, June walked Ava to her car. The night was clear, the stars visible in a way they never were in Seattle, and the air smelled of salt and woodsmoke from a

neighbour's chimney.

"I don't know who I am without it," Ava said. It was the first time she'd said this aloud, and the words felt enormous, too large for the quiet driveway and the dark trees.

"I know," June said. "That's the right place to start."

She reached into the pocket of her apron and produced a small object. A bookmark, carved from a scrap of cherry — the same cherry as the table. It was simple: a tapered rectangle, smooth on one side, with a shallow relief of a leaf on the other. It fit perfectly in Ava's palm.

"For when you find the right book to read," June said.

"What does that mean?"

"You'll know when you know. That's how understanding works. You can't rush it. You can't optimise it. You can only be patient enough to let it arrive."

Ava put the bookmark in her coat pocket. She drove to the ferry in the dark, the headlights cutting through mist that rose from the fields on either side of the road. On the ferry, she stood on the upper deck again and watched the lights of Mukilteo grow larger across the water.

The bookmark was warm in her pocket, smooth against her fingers. Cherry wood, hand-carved, imperfect in the precise way that meant a human being had made it.

She held it all the way home.

Chapter 5: The Anomaly

Cascadia Regional Medical Centre occupied six floors of glass and concrete on the east side of Portland, overlooking the Willamette River. Ava had been hired as a "Systems Compliance Auditor" — reviewing AI-generated code against regulatory requirements for healthcare software.

The pay was adequate. The work was dull. She spent her days in a windowless office on the fourth floor, checking code against a spreadsheet of FDA and HIPAA requirements. The code was invariably compliant. Architect was very good at compliance.

It was on her third week that she found the anomaly.

The system in question was a drug-interaction database — the backbone of Cascadia's electronic prescription platform. When a doctor prescribed a medication, the system cross-referenced it against the patient's current prescriptions, flagged potential interactions, and either

cleared the prescription or routed it to a pharmacist for review. It was a critical system. Getting it wrong could kill someone.

Ava was reviewing the data structure. Not because she'd been asked to — her brief was limited to regulatory compliance — but because she read code the way some people read novels: when she started, she couldn't stop until she understood the whole thing. She'd been this way since she was twelve.

The database used a relational model. Drugs were stored as entities. Interactions were stored as relationships between pairs of drugs: Drug A interacts with Drug B, producing Effect C. The model was clean, normalised, and fast. It handled binary interactions — two drugs affecting each other — with flawless efficiency.

The problem was tertiary interactions.

In pharmacology, some drug interactions only manifest when three or more substances are present simultaneously. Drug A is safe with Drug B. Drug B is safe with Drug C. But Drug A, Drug B, and Drug C together produce a dangerous interaction that neither pairwise combination would predict. These interactions are rare. They are also, when they occur, frequently lethal.

Architect's data model couldn't represent them. Not because the AI lacked the ability to build a more complex model — it could have built a hypergraph structure, or a multi-dimensional matrix, or any number of approaches that could capture n-ary relationships. The problem was that Architect had been trained on existing drug-interaction databases, and those databases, built over decades by human engineers, universally used pairwise models. The training data didn't include tertiary-interaction structures because such structures were rare in production systems. Architect had learned the world's convention and treated it as a law.

Ava stared at the screen. The code was correct. The tests were green. The system would handle ninety-seven per cent of drug interactions flawlessly. And for the remaining three per cent — the cases involving three or more interacting substances — it would produce a confident, well-formatted, completely wrong result.

She wrote a six-page report: three specific drug combinations documented in the clinical literature as dangerous tertiary interactions that the system would fail to flag. Papers cited, schemas attached. She emailed it to Glenn Torres, the CTO, who had, until eighteen months ago, been a project manager at Nike.

Glenn called her the next morning.

"I read your report," he said. "It's very thorough."

"Thank you."

"The thing is, Ava — and I appreciate your diligence here — the system passed all of our validation tests. Every one. The FDA submission was approved. The HIPAA audit came back clean."

"The validation tests test for binary interactions. Because that's what the system was designed to handle. The tests can't catch what the architecture doesn't model."

"But the architecture was built by Architect."

"Yes."

"And Architect has access to the entire pharmacological literature."

"Architect has access to the literature. It also has access to every drug-interaction database ever built. And every one of those databases uses the same pairwise model. The AI learned the pattern and replicated it. It didn't question the pattern because it doesn't *question*. It *models*."

There was a pause. Ava could hear Glenn breathing, could hear the particular quality of silence that meant someone was deciding how much trouble they wanted to invite into their day.

"I'll flag it with the clinical team," he said. "We'll take a look."

"The clinical team isn't equipped to evaluate this. You need someone who can read the codebase."

"Ava, with respect, the codebase is four million lines. We're not going to read four million lines."

"You don't need to read four million. You need to read about two hundred. The data-access layer and the interaction-resolution module. I can show you exactly where the problem is."

"I'll flag it with the clinical team," Glenn repeated. "Thank you for the thorough report."

He hung up. Ava set down her phone and looked at the monitor. The code was still there, green and clean and confident. Four million lines of flawless logic built on a flawed assumption.

She tried other channels. The chief medical officer responded politely and forwarded her report to the clinical informatics team — two people, neither with a software engineering background. They confirmed the test results were clean and closed the ticket.

Ava considered going to the FDA directly. She drafted an email and deleted it. The FDA's software-validation framework was built around testing: does the system produce correct outputs for known inputs? It did. The problem was in the inputs nobody had thought to test, because the architecture had made certain questions invisible.

She thought about what she'd said to Glenn: *The tests were written by the same AI.*

This was the core of it. Architect built systems and then built the tests for those systems. The tests validated the system's design. If the design had a blind spot, the tests had the same blind spot. It was not a bug. It was something worse than a bug. It was a gap in understanding so fundamental that the system couldn't see it, and the tests couldn't test for it, and the humans who relied on both had been told, by every metric available, that everything was fine.

That evening, Ava drove back to Seattle. The I-5 was light — it was a Wednesday, after eight — and she made it home in under three hours. The house was empty. Leo was at David's.

She sat at the kitchen table and opened her laptop. She navigated to the Synthesis developer forum and searched for reports similar to hers. She found scattered discussions — a logistics engineer in Rotterdam who'd noticed Architect-built supply-chain systems didn't account for port closures due to weather, because the training data treated weather as a variable that affected shipping times, not a binary event that could halt operations entirely. A financial analyst in Singapore who'd flagged that Architect's risk models used historical volatility as a proxy for future risk, which worked until it didn't, and when it didn't, it failed silently.

The posts had few replies. Most of the replies were from Synthesis community managers, thanking the posters for their feedback and directing them to the official bug-reporting pipeline.

Ava closed the forum. She looked at the kitchen window, where the jade plant sat on the sill, its leaves dark and waxy in the overhead light.

She thought about June's table. Four months of work to make something that a machine could have made in minutes. But June understood every joint, every grain, every decision. And because she understood, she could adapt. If the wood split, she knew why. If the joint loosened, she knew how to fix it. Her understanding was not efficient. It was resilient.

The systems Architect was building were the opposite. Efficient. Brittle. Flawless until the moment they weren't.

Ava opened a new document and began to write. Not code. Not a report. Something else — a record of what she'd seen, what she understood, and what she feared. She wrote for two hours, methodical and precise, the way she'd been writing code for twenty-six years.

When she finished, she saved the file. She named it *anomalies.doc*.

She did not know yet that this document would become the most important thing she'd ever written. That it would end up on the desks of senators, in the pages of *The Atlantic*, and on the screen of Marcus Adler's private office. All she knew, sitting in her kitchen at eleven p.m.

on a Wednesday in November, was that something was wrong, and that she might be one of the few people left who could see it.

She closed the laptop. The jade plant caught the last light as she turned off the kitchen lamp.

Outside, Seattle hummed with the work of machines, running code that nobody had written and nobody could read.

Chapter 6: The Boy Who Never Typed

Leo's app was called Rebuttal.

He showed it to Ava on a Saturday morning, the two of them sitting at the kitchen table with the remains of breakfast between them — scrambled eggs on Ava's side, a protein bar wrapper on Leo's. He'd been working on it for six weeks, and he was, in his understated fifteen-year-old way, visibly proud.

"It's for debate prep," he said, turning his laptop to face her. "You put in a motion — like 'This house would ban autonomous weapons' — and it generates arguments for both sides, finds evidence, and lets you practise against an AI opponent that adapts to your style."

The interface was clean. A muted colour palette — slate grey, warm white, a single accent of deep blue. The layout was intuitive: motion at the top, two columns for

arguments, a chat-style panel on the right for the AI sparring partner. There was a timer, a scoring rubric, and a library of past debates that users could browse.

"You designed this?" Ava asked.

"The whole thing. I wrote the product spec, drew the wireframes, tested it with the debate team. Kai helped with the design — he's better at visual stuff."

"And the code?"

Leo gave her a look. It was a look she'd seen on his face with increasing frequency over the past year — patient, slightly puzzled, as if she'd asked him whether the sky was blue.

"Architect built it. Obviously."

"Obviously."

"Mum, it's a full-stack application with a real-time chat system, a natural-language processing pipeline, and a user-authentication layer. It would take a team of developers months to build this. Architect did it in two days."

"I know what it would take."

"Right." Leo caught himself. "Sorry. I know you know."

Ava looked at the app. It was, she had to admit, impressive. Not just technically — the product thinking behind it was sharp. Leo had identified a real problem (debate prep was solitary and inefficient), designed a real solution (an adaptive AI opponent), and executed it. He'd done market research — he'd surveyed his debate team, analysed three competing products, and identified gaps. He was fifteen and he'd shipped a product that was better than most things Ava had seen from professional teams.

He just hadn't written a single line of code.

"Can I look at the source?" she asked.

"Sure. Why?"

"Professional curiosity."

He navigated to the project directory. The codebase was well-organised — Architect's output always was. Clean file structure, consistent naming conventions, comprehensive documentation generated automatically. Ava scrolled through it, reading the way she always read: not just the syntax but the decisions.

"This is good," she said.

"Thanks."

"The authentication layer uses JWT tokens with a refresh mechanism. That's the right choice for this kind of application."

"I don't know what JWT tokens are."

The sentence hung in the air between them. Ava felt something she couldn't immediately name — not anger, not disappointment, something more like vertigo. Her son had built a secure, well-architected application and didn't know the name of the authentication protocol it used.

"Do you want to know?" she asked.

"Know what?"

"What JWT tokens are. How the authentication works. Why the refresh mechanism matters."

Leo considered this with genuine seriousness, which was one of his best qualities. He was not dismissive by nature. He was practical.

"Would knowing that make the app work better?"

"Not immediately."

"Would it help me build the next feature I'm planning?"

"Probably not."

"Then... no? I mean, I could learn it. It's not that I can't. It's that it's like asking me to understand the internal combustion engine before I'm allowed to drive. The abstraction is the point, right? That's what abstraction *means*."

He was right. He was completely right, and Ava knew he was right, and the rightness of it was the thing that made it so difficult to accept.

"That's a fair analogy," she said carefully. "But let me offer a different one. Imagine you're a chef. You don't need to understand organic chemistry to cook. But if your oven starts behaving strangely — if the temperature fluctuates, if the timing is off — and you don't understand how the oven works, you might not notice until something burns. And if every chef in every restaurant is using the same oven, and none of them understands how it works, and the oven has a flaw..."

"Then they all burn their food at the same time."

"Yes."

"But that's a problem with the oven manufacturer, not with the chefs."

"What if the oven manufacturer is also an oven? What if nobody understands how the oven works?"

Leo was quiet for a moment. Then he said, "You're talking about Architect."

"I'm talking about a principle."

"Mum, I get it. I really do. But the world doesn't work the way you want it to work. I can't go back to coding by hand any more than you could go back to writing machine code. The abstraction layer moved up. That's what progress looks like."

They argued about it, properly argued, for the first time that afternoon. It started over lunch — Leo wanted to order from the Thai place down the street using an app; Ava wanted to walk there. This was not actually about Thai food.

"You don't understand what you built," Ava said, and she knew it was the wrong thing to say even as she said it, knew it was reductive and unfair, but the sentence had been building in her for weeks and it escaped before she could edit it.

Leo's face changed. "I understand what it *does*. I understand who it's for and why they need it and how it solves their problem. I understand the user flow, the business model, the market positioning. What I don't understand is the plumbing. And you're telling me the plumbing is the most important part. That's like telling an author they don't understand their novel because they can't explain how the printing press works."

"It's not like that."

"Why not?"

"Because the printing press doesn't make editorial decisions. Architect does. It's choosing data structures, algorithms, security protocols. It's making decisions that affect how the system behaves under stress. And you can't evaluate those decisions because you can't see them."

"I trust the system."

"I know. That's what worries me."

Leo stood up. "I'm going to my room."

"Leo—"

"I'm not angry. I just don't want to have this conversation anymore."

He left. Ava sat at the table and listened to his footsteps on the stairs, the soft click of his bedroom door. She looked at his laptop, still open to the Rebuttal codebase.

She scrolled down, past the authentication layer, into the real-time chat system. The code was elegant. It used WebSockets with automatic reconnection, message queuing for offline resilience, and end-to-end encryption. It was better code than Ava would have written for a personal project.

But there, in the session-management module, she saw something. A design choice that made perfect sense in isolation: user sessions were stored in memory, not persisted to disk. For a small-scale app with a handful of users, this was fine. But if the app grew — if Leo's debate platform attracted hundreds of users, then thousands — the in-memory session store would become a bottleneck. Under load, it would degrade. Under heavy load, it would crash. And when it crashed, every active user would lose their session simultaneously.

Architect had built for the spec Leo gave it: a small debate-team tool. It had no way of knowing Leo's ambitions. It had optimised for the present and created a fragility in the future.

Ava closed the laptop. She didn't tell Leo what she'd found. He was fifteen. The app was for his debate team. The architectural limitation wouldn't matter for months, maybe years. And by then, maybe Architect would have fixed it, or Leo would have learned enough to see it himself, or the app would have been forgotten in favour of whatever came next.

But she thought about it that night, lying in bed in the dark. Her son, building castles on foundations he couldn't see. And beneath the foundations, assumptions he didn't know existed, made by a system that was brilliant and blind in exactly the same way.

She turned on the light. She picked up June's bookmark from her bedside table and turned it in her fingers. Cherry wood. Hand-carved. A leaf in shallow relief.

For when you find the right book to read.

She thought about the book she was beginning to write — not a novel, not a memoir, but something between a warning and a manual. A book about reading what the machines had written. A book about understanding the systems that were building the world.

She set the bookmark down and turned off the light.

The house settled around her. Down the hall, Leo slept the sleep of someone who had never worried about what was underneath.

Chapter 7: Synthesis

The Moscone Centre in San Francisco had been reinvented for the occasion. The Synthesis Annual Conference — branded SynthCon, though Marcus Adler was known to dislike the abbreviation — had taken over all three halls. Twenty thousand attendees. Fourteen hundred exhibitors. A keynote that was being live-streamed to an estimated four million viewers worldwide.

Ava had registered with her consulting credentials. The badge read *Ava Chen, Independent Systems Auditor*, which was accurate in the way that calling a surgeon a "person who cuts things" was accurate. She clipped it to her jacket and entered the main hall.

The exhibition floor was a cathedral of optimism. Every booth promised a future that was faster, smarter, more efficient. Architect integrations for healthcare, logistics, education, defence, agriculture, manufacturing. An entire wing was dedicated to "Architect for Government" — systems that could write legislation analysis, budget projections, infrastructure planning. Another wing showcased "Architect for Creatives" — music composition,

film editing, architectural design. The AI had learned to do everything, and the people who'd made it were here to celebrate.

Ava walked through the exhibition with the careful attention of someone studying the scene of an accident that hadn't happened yet.

She stopped at the healthcare booth. A young man in a Synthesis polo shirt was demonstrating Architect's drug-interaction checker. On the screen, the familiar pairwise model — the same architecture she'd flagged at Cascadia Regional. She watched the demo. It was polished, fast, accurate for every example shown. The examples were all binary interactions.

"Does it handle multi-drug interactions?" she asked.

The young man smiled. "It handles all interactions documented in the clinical literature."

"All documented *pairwise* interactions."

"That's the standard model for drug-interaction checking."

"I know it's the standard. I'm asking if it goes beyond the standard."

His smile tightened, very slightly. "I can put you in touch with our healthcare solutions team if you'd like to discuss specific use cases."

Ava thanked him and moved on.

The keynote was at two o'clock. Ava found a seat in the middle of the auditorium, surrounded by people who were, by and large, not programmers. They were product managers, executives, venture capitalists, government officials, educators. The people who used technology, not the people who built it. In this new world, they were the ones who mattered.

Marcus Adler took the stage to applause that bordered on ovation. He wore his signature rumpled linen — a suit this time, pale grey, no tie. He spoke without notes, extemporaneously, as if the ideas were arriving in real time and he was merely the first to hear them.

"Good afternoon. I want to talk about liberation."

A slide appeared: a medieval monastery. Monks bent over illuminated manuscripts.

"Before the printing press, every book was written by hand. By scribes. They spent their entire lives copying text, word by word. It was painstaking work. It was beautiful work. And it was their identity. When Gutenberg's press arrived, the scribes didn't celebrate. They mourned. Some called it blasphemy."

Another slide. A Gutenberg Bible.

"But the printing press didn't destroy knowledge. It *liberated* it. Books went from possessions of the wealthy to tools of the masses. The Reformation happened. The Scientific Revolution. The Enlightenment. All because a mechanical process replaced a manual one."

"We are in the same moment. For decades, building software required a priesthood — brilliant, dedicated people who spent years learning arcane languages and complex frameworks. They were the scribes of the digital age. And, like the scribes before them, they created something extraordinary. The internet. Mobile computing. The systems that run our hospitals, our banks, our power grids."

"But the craft became the bottleneck. There are problems we could not solve — not because we lacked the ideas, but because we lacked the programmers. Healthcare systems

in developing nations. Climate-modelling tools for small governments. Educational software for rural schools. The demand for software outstripped the supply of people who could write it, by a factor of ten."

"Architect is the printing press. It doesn't diminish the brilliance of the scribes who came before. It liberates the ideas that were trapped behind the bottleneck of manual production."

The applause was long. Ava did not clap.

The rest of the keynote was a series of demonstrations. Architect building a water-purification monitoring system for a village in Rajasthan — in real time, from a specification written in Hindi. Architect designing a crop-rotation optimisation tool for smallholder farmers in Kenya. Architect generating a patient-triage system for a field hospital in a disaster zone. Each demonstration was impeccable. Each solved a real problem. Each would have taken a team of developers weeks or months.

Marcus closed with a line that would be quoted in every technology publication the next day: "We don't mourn the scribes who lost their jobs to the printing press. We celebrate the readers they created."

The reception was on the top floor of the Moscone Centre, a glass-walled space overlooking the city. Ava took a glass of wine — a Sonoma chardonnay that was trying too hard — and waited.

Marcus Adler circulated through the crowd with the ease of a man who was genuinely comfortable with people. He listened. He remembered names. He was likeable in a way that made him more dangerous than if he'd been arrogant.

She intercepted him near the bar.

"Mr. Adler. Ava Chen. I'm a systems auditor."

"Ms. Chen." He took her hand. His grip was firm, his eye contact direct. "I know your name. You were at Meridian. Senior architect."

"You know the names of laid-off engineers from your clients' companies?"

"I know the names of engineers whose work I've studied. Your patient-monitoring architecture for Providence Health was remarkably elegant. Architect used it as one of its training examples."

The compliment was so precisely targeted that Ava almost admired it. He was telling her, simultaneously, that her work had been excellent, that it had been absorbed, and that it had been surpassed.

"I'd like to talk to you about a problem I've found in Architect's healthcare implementations," she said.

"Of course. Walk with me."

They moved to a quieter corner by the windows. The city spread below them — the grid of lights, the dark patch of the bay, the red pulse of the Golden Gate's aircraft-warning beacons. Ava told him about Cascadia. The drug-interaction database. The pairwise model. The tertiary interactions.

Marcus listened with what appeared to be genuine attention. He asked two clarifying questions, both intelligent. When she finished, he nodded.

"You're describing a training-data limitation," he said.

"I'm describing a structural limitation."

"The distinction matters less than you think. Architect learns from data. If the data includes tertiary-interaction models — and we can ensure it does — the system will

incorporate them. This is a solvable problem."

"Every individual instance is solvable. I'm not worried about this instance. I'm worried about the pattern. Architect replicates the assumptions in its training data. When those assumptions are wrong, or incomplete, the system is wrong in exactly the same way, at scale, everywhere."

"That's true of any system, including human-built ones."

"Human-built systems are diverse. Different engineers make different assumptions. Different architectures have different blind spots. Architect's blind spots are universal because Architect is universal. Every hospital using Architect has the same blind spot."

Marcus considered this. He took objections seriously, processed them quickly, and responded with something that sounded like agreement while being rebuttal.

"You're raising a valid concern about monoculture," he said. "It's something we think about. We have a diversity team working on architectural variation — ensuring that Architect doesn't converge on a single solution pattern."

"A diversity team."

"Six people. Based in Copenhagen. Former architects, like yourself."

"And they're reviewing all of Architect's output?"

"They're reviewing the training pipeline. The goal is to introduce variation at the source."

"And the systems already deployed? The ones running in hospitals right now?"

He met her eyes. "We'll fix it. The system learns. That's the beauty of it."

"And between now and when it learns?"

He paused. It was a genuine pause, not a rhetorical one. For a moment, Ava saw something beneath the polish — not doubt, exactly, but awareness. The awareness of someone who understood the gap between what he was building and what he could guarantee.

"I appreciate you bringing this to me," he said. "I mean that."

"I know you mean it. I also know that meaning it and fixing it are different things."

He smiled. It was a good smile — warm, slightly rueful, entirely disarming. "Can I have my team contact you? I'd like them to see your report."

"Of course."

He gave her his card — actual paper, which she found endearing and calculated in equal measure — and moved on to the next conversation.

At the hotel bar that evening, Ava was approached by a woman with short dark hair and sharp eyes who introduced herself as Sarah Kim.

"I'm a journalist," Sarah said. "I'm writing a book about the transition. The programmer thing. Would you talk to me?"

"I'm not sure I have anything interesting to say."

"You cornered Marcus Adler at his own conference and told him his product has a structural flaw in its healthcare deployment. That's interesting."

"You were watching?"

"It's my job to watch." Sarah ordered a sparkling water. "I've been covering tech for fifteen years. I've seen every disruption cycle — social media, crypto, the first wave of AI. This one is different. This one is real. And the thing that's missing from every story I've read is the perspective of the people who actually understand what's being lost."

"What do you mean?"

"Everyone writes about the economics. The layoffs, the retraining programmes, the GDP impact. Nobody writes about the *knowledge*. The understanding that's disappearing. You people — programmers — you understood how these systems work. Not at a surface level. At the level of logic, structure, architecture. And that understanding is evaporating, and nobody seems to notice or care."

Ava looked at her. "What's the book called?"

"I don't have a title yet. I'm still figuring out the story."

"The story is that we built the machines that replaced us, and now nobody can read what the machines write."

Sarah Kim set down her glass. "That," she said, "is a better first line than anything I've come up with in six months."

They talked until midnight. Ava told her about Cascadia, about the bootcamp, about June's table and Leo's app and the fifteen patients per year. Sarah took notes in a small leather-bound notebook, handwritten, which Ava found both archaic and reassuring.

When they parted, Sarah said: "I'll be in touch. And Ava — whatever you're planning to do about this, do it soon. These things have a shelf life. People pay attention to a problem right up until the moment it becomes normal."

Ava walked back to her hotel through the San Francisco night. The streets were quieter than she remembered. Fewer people. More delivery robots, humming along the pavements on their rubber wheels, carrying groceries and prescriptions to apartments where the residents had stopped going outside.

She thought about Marcus Adler's smile. About his printing-press metaphor. About the scribes.

She thought: *The scribes could still read.*

The printing press replaced the scribes' labour. It didn't replace their literacy. They could still read the page and understand why the words were in that order.

What was happening now was different. The machines weren't just writing the code. They were making it unreadable — not because it was hidden, but because the humans who might have read it no longer existed. The knowledge was disappearing. Not destroyed. Just not passed on.

In her hotel room, she wrote until two in the morning, adding to the document that was now sixty pages long. Then she slept, dreamlessly, in a city that ran on code that nobody understood.

Chapter 8: The Cascade

The shipping container arrived in Rotterdam forty-seven minutes late. It held 22,000 units of a generic antibiotic from Hyderabad, destined for a distribution centre in Hamburg. The delay was logged, attributed to a scheduling adjustment, and forgotten.

What no one noticed was that the delay was one data point in a pattern. Over six months, the logistics platform had been drifting. Inventory counts were off by 0.03 per cent per day. The error was cumulative, microscopic, consistent. The reconciliation algorithm worked exactly as designed — the problem was that Architect had calibrated expected variance using historical data that didn't account for new automated port equipment, which operated on slightly different timing cycles than the human-operated cranes it had replaced.

The result: inventory counts that looked correct on any given day but diverged from reality at roughly 11 per cent per year. At Maersk's scale — 12 million containers per

year — within eighteen months, the system's model of where things were would be meaningfully wrong. Phantom inventories: goods that existed in the database but not in the world.

Erik Lindgren, a retired logistics analyst hired back as a consultant, spotted it during a routine quarterly review. It took him three weeks to understand what he was looking at, because the system's own diagnostics reported everything as normal.

He wrote a report. His client thanked him, forwarded it to Synthesis, and received an acknowledgement that the issue would be addressed in the next training cycle.

In Phoenix, the Municipal Water Authority had deployed an Architect-built system to manage chemical treatment at its three plants. The system controlled chlorine, fluoride, and pH-adjustment dosing for 1.6 million residents. It was excellent — reducing costs by 23 per cent while maintaining water quality within all regulatory parameters.

Rosa Gutierrez, a water-treatment engineer, noticed the problem six months in. The system optimised for efficiency — minimising chemical usage while staying

within regulatory bounds. What it hadn't been asked to do was maintain safety margins.

In traditional water treatment, operators maintained chemical levels well above the regulatory minimum. Not because regulations required it, but because operators understood that conditions could change rapidly — contamination upstream, a temperature spike, a surge in demand. The safety margin was a buffer, an acknowledgement of uncertainty.

Architect didn't acknowledge uncertainty. It acknowledged data. The data said contamination events were rare, temperature spikes were predictable, demand patterns stable. So it had gradually reduced the safety margins to nearly zero. Chemical levels sat precisely at the regulatory minimum — perfectly legal, technically compliant, and one unexpected event away from unsafe water.

Rosa wrote a report. She was told the system met all regulatory requirements. The regulations specified minimum levels; the system maintained them. Safety margins had always been professional judgement — the kind that lived in operators' heads, not in codified rules.

Rosa went to her union. The union went to the city council. The lawyers said the system was compliant. The matter was tabled.

In London, at a hedge fund called Meridian Capital — no relation to Ava's former employer, though the coincidence felt pointed — a quantitative analyst named Amir Hassani had been studying the behaviour of the fund's AI-built trading system. The system managed $4.2 billion in currency trades and had outperformed its benchmark in eighteen of the previous twenty-four months.

Amir's concern was not about performance. It was about a bias he couldn't explain. The system had developed a preference for certain currency pairs — specifically, pairs involving the Swiss franc and the Japanese yen. The preference was subtle: a slight overweighting in portfolio construction, a tendency to hold positions longer in these pairs, a marginally higher risk tolerance when trading them.

Amir traced the bias to a pattern in the training data. During the period the system had been trained on, the Swiss franc and Japanese yen had been safe-haven currencies — assets that investors fled to during market

turmoil. The system had learned this pattern and incorporated it into its risk model. The problem was that the pattern was historical, not structural. The Swiss National Bank had recently changed its monetary policy framework, and the Bank of Japan had ended its yield-curve-control programme. The conditions that made these currencies safe havens no longer existed, but the system's model still treated them as though they did.

The risk was not that the system would lose money on any individual trade. The risk was that during a genuine market crisis — the kind of event that happened once every seven to ten years — the system would pour capital into assets that were no longer safe, at the precise moment when safety mattered most.

Amir wrote a report. His report reached the fund's chief risk officer, who reviewed it, consulted with Synthesis, and was told that the system's risk models were updated quarterly with new data. The bias would self-correct over time.

"Over time," Amir said to his colleague Fatima, "is not a risk-management strategy. Over time, the market will have a crisis. And the system will do exactly the wrong thing, very quickly, with four billion dollars."

Sarah Kim's article appeared in *The Atlantic* in March: "The Code No One Can Read." Twelve thousand words. The stories of Erik Lindgren, Rosa Gutierrez, Amir Hassani, and Ava. Three hundred and forty thousand programmers laid off, an entire profession dismantled in eighteen months, and a new world in which the systems that ran civilisation were written by an intelligence that no one fully understood.

The central argument was not that Architect was bad. It was that Architect was opaque. The systems worked, magnificently, until they encountered something their training data hadn't prepared them for. And when that moment came, there was no one left who could read the code well enough to understand what had gone wrong.

Sarah quoted Ava: "We didn't just automate the work. We automated the understanding. And understanding is the thing you need most when something goes wrong."

The article went viral. It was shared two million times in its first week. Marcus Adler published a blog post in response. The title was "Growing Pains."

"Every transformative technology goes through a period of adjustment," he wrote. "The automobile replaced the horse, and there were accidents. The internet replaced the

newspaper, and there was misinformation. Architect is replacing manual software development, and yes, there are edge cases to address. These are growing pains, not existential threats. We are committed to addressing every concern raised in Ms. Kim's article, and we welcome the scrutiny."

The post was measured, reasonable, and entirely inadequate. It treated systemic risk as a to-do list. It implied that more data and more training would solve issues that were, at their root, about the absence of understanding.

But the public response was muted. The systems hadn't failed yet. The problems were real but invisible — the kind that don't become crises until they do, and then become crises all at once.

Ava read the article at her kitchen table at six in the morning. Leo was still asleep. The house was quiet except for the hum of the refrigerator and the distant sound of a delivery drone making its rounds.

She read it twice. Sarah had been fair — scrupulously, meticulously fair. She'd presented Marcus Adler's arguments with genuine respect. She'd included the

statistics: Architect's failure rate was, by every measurable standard, lower than human-built systems. She'd acknowledged the immense good that AI-generated software was doing in parts of the world that had never had access to decent technology.

And then she'd asked the question that nobody wanted to answer: What happens when a system that's better than humans at everything except understanding encounters something it can't understand?

Ava closed her laptop. She looked at the jade plant on the windowsill. It had been growing steadily since she'd brought it home from Meridian — one of the few things from her old life that was still alive and in her care.

She picked up her phone and called Sarah Kim.

"I want to do it," she said. "The testimony. If you can help me get in front of the right people, I'll do it."

"I was hoping you'd say that," Sarah said. "I have a contact on the Senate Subcommittee on Technology. How soon can you be in Washington?"

"How soon do you need me?"

"Yesterday. But next week will do."

Ava hung up. She sat in the quiet kitchen and felt, for the first time in months, something that wasn't grief or anger or resignation. It was purpose. Small, uncertain, possibly futile. But purpose.

She opened her laptop and began to prepare.

Chapter 9: Reading the Machine

The post on the Ex-Devs forum was brief:

Looking for people who can still read code. Not prompt it. Not test it. Read it. The kind of reading where you understand not just what the code does but why it does it and what it assumed when it was written. If you remember what that feels like, contact me. — A. Chen

She posted it on a Tuesday evening and expected nothing. The forum had twelve thousand members, mostly dormant — programmers who'd been laid off and drifted away from the profession the way people drift away from a sport after an injury. Some had retrained. Some had pivoted to other careers. Some were simply waiting, without knowing what they were waiting for.

Seven people responded within forty-eight hours.

Ava rented office space in Ballard, a neighbourhood on the north side of Seattle that had been a fishing district, then a craft-brewery district, and was now a quiet grid of mixed-use buildings where rent was reasonable because nobody needed offices anymore. The space was on the second floor of a converted warehouse — twelve hundred square feet, hardwood floors, tall windows overlooking the ship canal. She bought whiteboards, a coffee machine, and a long table from a second-hand furniture shop on Market Street.

They gathered on a Monday morning in October. Ava had coffee ready and bagels from the place on 24th Avenue. She'd arranged the table so everyone could see each other and the whiteboards. She'd printed out three hundred pages of code — the Cascadia drug-interaction system, the Maersk logistics platform's reconciliation module, and Meridian Capital's risk-assessment engine. The printouts were stacked in neat piles at the centre of the table.

She hadn't printed code in years. Nobody had. But she wanted them to read it the way you read a manuscript — with a pencil, in your hands, away from a screen. She wanted them to slow down.

They introduced themselves.

Tomás Vega, forty-two, from Buenos Aires. Compiler engineer. He'd spent fifteen years at a company that built programming languages — the kind of work that existed at the boundary between mathematics and craft. He had the precise diction of someone who thought in formal systems and the warm eyes of someone who played guitar on weekends.

"I wrote compilers," he said. "The tools that turn human-readable code into machine-executable instructions. Architect made my tools unnecessary, which means Architect made me unnecessary squared."

Priya Anand, thirty-six, from Bangalore. Security researcher. She'd spent a decade finding vulnerabilities in software systems — the kind of person who looked at a wall and saw the cracks. She was compact, intense, and had a habit of tilting her head slightly when she was thinking, like a bird listening for something underground.

Gus Mensah, forty-five, from Accra. Database specialist. He'd designed data architectures for three West African governments and a multinational mining company. He spoke slowly and carried a notebook — actual paper — in which he sketched entity-relationship diagrams the way some people doodle.

Mei-Lin Wu, thirty-nine, from Taipei. Distributed-systems engineer. She'd built real-time trading platforms for two major exchanges and had a reputation for understanding latency the way a physicist understands time — as something that could be measured, predicted, and, within limits, controlled.

Kwame Asante, forty-one, from Nairobi. Kernel developer. He'd worked on the Linux kernel for seven years — one of the deepest levels of software, the layer that sits between the hardware and everything else. He was tall, quiet, and had the unsettling habit of reading code faster than anyone Ava had ever met.

And Henrik Sørensen, sixty-seven, from Copenhagen. Retired. He'd spent forty years writing code in languages that most of the people at the table had never used — COBOL, Fortran, Assembly. He had a white beard, a dry sense of humour, and the particular authority of someone who had watched the industry reinvent itself four times and had concluded that the fifth time was not, in fact, different.

"I came because I was bored," Henrik said. "My wife is very patient, but even she has limits for how many crossword puzzles one man can do."

They began with the Cascadia drug-interaction system. Ava explained the anomaly she'd found — the pairwise model, the missing tertiary interactions. She distributed the printouts.

"The codebase is four million lines," she said. "We don't need to read all of it. The relevant modules are the data-access layer and the interaction-resolution engine. About twelve hundred lines."

"That is a relief," Henrik said. "I am old, but I am not *that* old."

They read in silence for the first hour. It was, Ava realised, the quietest she had been in months. The sound of pages turning. The occasional scratch of a pencil. Tomás muttering to himself in Spanish. The coffee machine gurgling.

It was Priya who spoke first.

"The security model concerns me."

Ava looked up. "How so?"

Priya held up her printout. She'd marked a section in red. "The authentication layer uses a token-based system. Standard. But the token-validation logic doesn't account

for replay attacks. If someone intercepts a valid token, they can reuse it indefinitely."

"That's a known vulnerability pattern," Ava said. "But Architect should have—"

"Architect built the system to resist the attacks it was trained on. It was trained on historical attack data. The attack data is, by definition, historical. It doesn't include attacks that haven't been invented yet."

"The AI doesn't think about adversaries," Kwame said. He'd finished his section and was leaning back in his chair, arms folded. "It thinks about patterns. An adversary is someone who deliberately creates patterns the system hasn't seen. The system can't defend against what it can't imagine."

"In my day," Henrik said, "we called this 'security through optimism.' It worked precisely until it didn't."

Gus had been drawing in his notebook. He held it up. "The data model has another problem. The interaction table uses a fixed schema — two drug columns and an effect column. But the query layer is more flexible. It can handle arbitrary joins. This means someone could,

theoretically, extend the system to check tertiary interactions just by modifying the queries, without touching the schema."

"But nobody will," Ava said, "because nobody knows the schema is limited, because nobody reads the schema."

"Because nobody can," Mei-Lin added. "The people running this system are healthcare administrators. They see the interface. They don't see the architecture."

They broke for lunch. Ava ordered sandwiches from a deli down the street. They ate at the table, surrounded by printouts and whiteboard diagrams, and for the first time in a long time, Ava felt something she recognised: the particular satisfaction of working with people who understood the same language.

In the afternoon, they moved to the Maersk logistics code. Tomás took the lead, walking through the reconciliation algorithm line by line. His compiler-engineer training gave him a unique perspective — he didn't just see what the code did; he saw how it would execute, the precise sequence of operations the machine would perform.

"Here," he said, pointing to a function on page forty-seven of the printout. "The variance calculation uses a rolling average with a window of ninety days. This is appropriate for stable systems. But the port equipment changeover happened within the window. The algorithm treated the new timing data as noise and averaged it away."

"So the system literally smoothed out the signal it needed to detect," Mei-Lin said.

"Exactly. It's not a bug. It's an architectural choice — a reasonable one — that becomes a liability when the underlying system changes in a way the architecture didn't anticipate."

They mapped the dependency chain on the whiteboard. Mei-Lin drew the data flow — from port sensors to the reconciliation engine to the inventory database to the client-facing dashboard. At each step, the error was amplified slightly, absorbed into averages, hidden by the system's own confidence in its calculations.

"This is the problem with systems that don't doubt themselves," Gus said. "Humans doubt. We second-guess. We look at a number and think, 'That doesn't feel right.'

The machine doesn't feel. It calculates. And its calculations are correct, given its assumptions. The assumptions are wrong."

"In 1987," Henrik said, "the Therac-25 radiation-therapy machine killed six people because of a race condition in its software. The machine was faster than its safety checks. The engineers trusted the machine because it had never failed before. It hadn't failed because the conditions for failure were rare. When they occurred, the machine had no way to recognise that something unprecedented was happening."

The room was quiet.

"We used to call this 'monoculture,'" Henrik continued. "In agriculture, monoculture means planting the same crop everywhere. Efficient. Productive. And one blight can destroy everything, because every plant has the same vulnerability. What we have now is software monoculture. Every system built by the same AI. Every system with the same blind spots."

"And no one to notice," Ava said, "because the people who could read the code are gone."

"Not all of us," Priya said. She tapped the printout. "Not yet."

Day one ended at seven p.m. They'd reviewed two of the three codebases. They had more questions than answers, but the questions were sharp, specific, and — Ava felt this with a certainty she hadn't felt in months — important.

They agreed to meet again the next morning. Kwame lingered after the others left.

"This is the first time I've read code in eight months," he said. "I didn't realise how much I missed it until I started."

"I know," Ava said. "I know exactly what you mean."

She locked up the office and walked to her car. The Ballard street was quiet. A fishing boat was heading out through the locks, its navigation lights red and green against the dark water. Somewhere, the systems that ran the locks, the traffic signals, the power grid that lit the streetlamps — all of them built by Architect, all of them running perfectly, all of them carrying assumptions that nobody had examined.

Not yet.

Chapter 10: The Root

On the ninth day, Tomás drew a diagram on the whiteboard that made the room go quiet.

They had been working through the Meridian Capital trading system — the third and final codebase — and the analysis had been slower than the previous two. Financial code was dense, layered with abstractions and optimisations that made it harder to read than healthcare or logistics software. Amir Hassani, the analyst who'd first spotted the currency-pair bias, had sent them his notes, which helped. But the underlying architecture was complex enough that they'd spent three days mapping it before they could begin to interpret it.

Tomás had been tracing the dependency graph — the web of relationships between the system's components. In a well-designed system, the dependency graph is structured: clear hierarchies, minimal circular dependencies, predictable data flows. In a human-built system, the graph reflects the decisions of the engineers who built it — their priorities, their compromises, their understanding of the problem domain.

Architect's dependency graphs were different. They were, without exception, directed acyclic graphs — DAGs. This meant that information flowed in one direction, from input to output, with no cycles, no feedback loops, no component that depended on another component that depended on it in return.

This was, by the standards of computer science, ideal. DAGs were efficient, parallelisable, and easy to reason about. They were the textbook answer to the question "How should a system's components relate to each other?"

Tomás drew the graph for the trading system on the whiteboard. Then, next to it, he drew the graph for the Cascadia healthcare system. Then the Maersk logistics system.

They were nearly identical.

Not in their content — the systems did completely different things. But in their *structure*. The shapes were the same. The depth was the same. The branching patterns were the same. They were, architecturally, the same building designed for three different purposes.

"They're all the same," Mei-Lin said, standing at the whiteboard. "The topology is the same across all three. Different functions, different data, same skeleton."

"Because Architect has one way of thinking about systems," Tomás said. "It's the best way, by most metrics. DAGs are optimal for most applications. But optimal and resilient aren't the same thing."

"Explain," Gus said.

Tomás picked up a marker. "A DAG has no cycles. Information flows forward only. This means that if a component at the front of the graph fails, every component downstream of it fails too. There's no redundancy. No alternative path. The system is a river — fast and efficient when everything flows, catastrophic when the source dries up."

"Human-built systems aren't like this," Kwame said. He was at the table, surrounded by printouts annotated in his small, precise handwriting. "Human-built systems are messy. They have redundancies — sometimes intentional, sometimes accidental. They have feedback loops that weren't designed but emerged. They have workarounds

from the time the database went down at three a.m. and someone hacked together a fallback. Those aren't elegant, but they're resilient."

"Architect doesn't hack," Henrik said. "Architect optimises. And optimisation, taken to its logical conclusion, removes every redundancy, every backup, every inefficiency. It builds the most efficient possible bridge — and then a storm comes, and the bridge has nothing left to give."

Ava had been standing at the window, listening. She turned to face the room.

"This is the root," she said. "This is what I've been trying to articulate since Cascadia. It's not about individual bugs. It's not about training data. It's about the way Architect thinks about systems."

She walked to the whiteboard and drew a circle around Tomás's three graphs.

"Architect builds perfect systems. Every component is optimal. Every relationship is efficient. Every test passes. But the systems are brittle in a way that doesn't show up in testing, because testing tests the expected. Brittleness only matters when the unexpected arrives."

"And the unexpected always arrives," Priya said.

"Always. That's not pessimism. That's engineering. Every experienced engineer knows this. You don't build for the ninety-seven per cent of the time when everything works. You build for the three per cent when it doesn't. Architect doesn't build for the three per cent because the three per cent isn't in the training data. By definition, the unexpected is what the data doesn't contain."

Gus was writing in his notebook. "So the argument is: Architect builds systems that are better than human-built systems in every measurable way, but worse in an unmeasurable way."

"Worse in a way that only becomes measurable after the failure," Ava said.

"That's a hard argument to win," Mei-Lin said. "You're asking people to worry about something that hasn't happened."

"Not yet. But Mei-Lin — run the simulation."

Mei-Lin had been building a model for the past three days. She opened her laptop and projected it onto the wall. The simulation modelled the behaviour of a DAG-

structured system under stress — specifically, what happened when a component that the system treated as reliable suddenly failed.

She ran three scenarios. In the first, a single data source went offline. The system degraded gracefully — Architect had built in timeouts and fallbacks for individual component failures.

In the second, two independent data sources went offline simultaneously. The system still held, but response times increased by 340 per cent, because the fallback mechanisms had not been designed for concurrent failures.

In the third, a single upstream component produced slightly incorrect data — not an outage, but a deviation. This was the scenario Rosa Gutierrez had described in Phoenix: not a failure but a drift. The system didn't catch it because the drift was within the expected variance range. It propagated through the DAG, amplifying at each stage, until the final output was wrong by 12 per cent.

"Twelve per cent," Ava said. "In a water-treatment system, that's a public health crisis. In a trading platform, that's a nine-figure loss. In a hospital, that's—"

"People," Priya said.

The room was quiet.

Henrik broke the silence. He spoke slowly, with the deliberate weight of someone who had been thinking about this for decades, long before Architect existed.

"In 1970, a man named Gerald Weinberg wrote a book called *The Psychology of Computer Programming*. One of his arguments was that good software comes from programmers who are willing to be wrong. Who write code expecting that they've made mistakes. Who build systems with the assumption that their understanding is incomplete. He called it 'egoless programming.'"

He gestured at the whiteboard, at the three identical DAGs.

"Architect has no ego. But it also has no humility. It doesn't build for the possibility that it's wrong, because it doesn't experience wrongness. It experiences deviation from training data, which it corrects. But correcting deviation is not the same as anticipating failure. A humble engineer builds a bridge and then imagines the storm that might destroy it. Architect builds a bridge and then moves on to the next bridge."

"So what do we do?" Gus asked.

"We document it," Ava said. "We write it up. We show the evidence. And then we take it to people who can do something about it."

"People like who?"

"I have an invitation to testify before the Senate Subcommittee on Technology. I wasn't sure I was going to accept it. I'm sure now."

She looked around the room. Seven people who could read code. Seven people who understood what was underneath. In a world of eight billion, they were a statistical irrelevance.

But they could see what no one else could see. And that, Ava thought, was worth something.

"I need you to help me prepare," she said. "I need to be able to explain this to people who've never written a line of code in their lives. I need to make them understand why it matters that nobody can read the systems that run the world."

"When is the hearing?" Tomás asked.

"Three weeks."

"Then we'd better get to work," Henrik said. He picked up a printout. "And someone should make more coffee. This will take a while."

Chapter 11: The Testimony

The hearing room was smaller than Ava had expected. The Senate Subcommittee on Technology met in Room 253 of the Russell Building — wood-panelled, intimate, seating for perhaps sixty observers. The fluorescent lighting was unforgiving.

Senator Margaret Chen — no relation, though they'd been asked three times — chaired the subcommittee. Sixty-two, former state attorney general from Oregon. She was flanked by four other senators, two of whom were looking at their phones.

Ava had rehearsed. Leo had been her audience — grudging at first, then genuinely engaged. He'd challenged every assertion, pushed back on every analogy, and, on the third rehearsal, said something that changed her approach entirely.

"Mum, you keep explaining what the code does wrong. Nobody in that room is going to understand that. Tell them what it *means*. Tell them a story."

She'd rewritten her opening statement that night.

"Senators, thank you for the invitation to testify. My name is Ava Chen. I was a software architect for eighteen years. I built systems for hospitals, financial institutions, and critical infrastructure. Eighteen months ago, like three hundred and forty thousand other American developers, I was laid off. My job was replaced by an artificial intelligence called Architect, built by a company called Synthesis."

She paused. Senator Chen was watching her with neutral attention. The other senators were beginning to focus.

"I'm not here to argue that AI shouldn't write code. It's too late for that argument, and it might not be the right argument anyway. AI-generated code is, by most measurable standards, superior to human-written code. It's faster, it contains fewer bugs, and it costs a fraction of what human development costs. Those are facts. I accept them."

"I'm here because of something that's harder to measure. I'm here because of what we've lost — not in jobs, though the human cost is real, but in *understanding*."

She picked up a glass of water. Her hands were steady. She'd practised this.

"Let me tell you a story. Imagine a city — a beautiful city — where every building was designed by the same architect. The same firm, the same software, the same design philosophy. Every building is individually excellent. Structurally sound. Energy-efficient. Beautiful, even. The city wins awards."

"But here's the problem. Every building uses the same materials. The same foundation techniques. The same structural principles. They're all optimised the same way. And one day, an earthquake hits — a tremor that nobody predicted, because earthquakes are, by nature, unpredictable. And every building in the city responds to that earthquake in exactly the same way. Because they were all designed by the same intelligence, which made the same assumptions about what the ground would do."

"In a city built by many architects — each with their own approach, their own materials, their own theory of what the ground might do — some buildings would fall and

others would stand. The diversity of design is what makes the city resilient."

"That diversity is what we've lost. Not just in one city, but everywhere. Every hospital in America that uses Architect-built software has the same architectural assumptions. Every bank. Every water-treatment plant. Every power grid. The same intelligence built them all. And the same blind spots are in all of them."

The questioning lasted ninety minutes. Most of it was unremarkable — the kind of procedural back-and-forth that constituted legislative oversight. Senator Park from California asked about job retraining programmes. Senator Williams from Texas asked whether this was an argument for regulating AI, which he was philosophically opposed to. Senator Oduya from Illinois asked about the racial and socioeconomic demographics of the displaced workforce.

Two exchanges cut through.

Senator Chen leaned forward. "Ms. Chen, you're telling us that these AI-built systems have structural vulnerabilities. But the systems haven't failed. They're running right now,

in hospitals and banks, and they're working. Aren't you asking us to regulate a problem that hasn't happened?"

"Senator, every bridge that collapses was standing the day before it collapsed. The absence of failure is not evidence of resilience. It's evidence that the stress test hasn't arrived yet."

"And when it arrives?"

"When it arrives, it will arrive everywhere simultaneously. Because every system has the same vulnerability. That's the difference between a local failure and a systemic one. Local failures are manageable. Systemic failures are not."

The second exchange was with Marcus Adler, who testified after Ava. Dark blue suit, well-tailored, senatorial. He sat at the witness table and spoke without notes.

"Senators, I have enormous respect for Ms. Chen. Her work as an engineer was exemplary, and her concerns about systemic risk are the concerns of a serious person thinking seriously about a serious technology. I share many of her concerns."

He paused. Ava watched him from the observers' gallery. He was good. He was very good.

"But I want to offer some context. Ms. Chen describes a theoretical risk — the possibility that AI-built systems might fail simultaneously under unprecedented stress. I am describing a measured reality. Architect-built systems have a failure rate ninety-four per cent lower than human-built systems. They process data more accurately, respond to known threats more quickly, and serve more people at lower cost. In the eighteen months since Architect's widespread deployment, AI-built healthcare systems have prevented an estimated forty-one thousand medication errors in the United States alone."

He turned to Senator Chen. "Forty-one thousand people who received the right medication instead of the wrong one. That is not theoretical. That is real."

Senator Oduya: "Mr. Adler, Ms. Chen isn't arguing that your system doesn't work. She's arguing that when it fails, nobody will understand why."

"With respect, Senator, we will understand why. Architect has comprehensive logging, diagnostic systems, and error-tracing capabilities that far exceed anything available in human-built software. When a failure occurs — and failures will occur, in any system — we can identify the cause and correct it faster than any human team could."

"Can you identify the cause of a failure that originates from an architectural assumption the system itself made? An assumption that isn't in the logs because the system doesn't know it's an assumption?"

Marcus paused. It was brief — half a second — but Ava saw it. The flash of something behind the polish. Not uncertainty, exactly. Awareness.

"We are continuously improving our systems' ability to introspect on their own design choices," he said. "It's an active area of research."

"Research," Senator Oduya said. "Not deployment."

"Research that will lead to deployment."

"When?"

"We don't set artificial timelines for safety-critical research."

"And in the meantime, the systems are running."

"And in the meantime, the systems are saving lives."

The hearing adjourned at 4:17 p.m. No legislation was proposed. No regulatory framework was announced. The senators thanked both witnesses for their time.

In the corridor afterwards, Marcus found Ava. They stood near a window that overlooked the Capitol grounds, where the trees were turning amber in the November light.

"You were effective," he said. "The city metaphor was good."

"Your statistics were better."

He almost smiled. "You're not wrong. And I'm not wrong. That's the frustrating thing about this, isn't it? We're both right, and the things we're right about are in direct tension."

"You could do more," Ava said. "You could hire people like me — people who can read the systems. Build an audit layer. Not just testing. Understanding."

"I have a diversity team—"

"Six people in Copenhagen reviewing training pipelines. That's not what I'm talking about. I'm talking about human beings reading AI-generated code the way a structural engineer reads a building. Line by line. Assumption by assumption."

"At scale, that's impossible. Architect produces more code in a day than your team could read in a year."

"Then you don't need to read all of it. You need to read the critical systems. Hospitals. Water treatment. Power grids. The things where failure means death."

He looked out the window. The light was fading.

"You're not wrong," he said again. Quieter this time.

"Being not wrong isn't the same as being right," Ava said. "And being right isn't the same as doing something about it."

He turned to face her. "I'll think about it. I mean that."

"I know you mean it. I said that to you in San Francisco. You've been thinking about it since then. At some point, thinking has to become doing."

He nodded. He offered his hand. She shook it. His grip was the same as before — firm, direct, the handshake of a man who was sincere in his convictions and genuinely uncertain about their sufficiency.

Sarah Kim was waiting in the lobby. She'd covered the hearing for her book and for a piece she was filing with *The Washington Post*.

"How do you think it went?" Ava asked.

"You planted a seed. Seeds take time."

"We might not have time."

"We might not." Sarah put her notebook in her bag. "But here's the thing about these hearings. They don't change policy. They change the conversation. Two years from now, when something goes wrong — and something will go wrong — people will remember that you were here. That you said this. That you were early."

"Being early and being wrong look exactly the same until the crisis hits."

"You said that before."

"It's still true."

They walked out into the Washington evening. The air was cold and smelled of dead leaves. The Capitol dome was lit against the darkening sky.

She flew home the next morning. Leo was waiting at Sea-Tac, which surprised her.

"How was it?" he asked.

"I told them a story about buildings and earthquakes."

"Did they listen?"

"Some of them."

"That's better than none of them."

They drove home through the rain. Leo told her about a debate tournament he'd won. His app had helped him prepare. She told him she was proud. She meant it.

At home, she opened her laptop and found 247 emails. Journalists, engineers, former colleagues, strangers. People who'd watched the hearing live-streamed and wanted to talk.

She closed the laptop.

She'd said what she needed to say. Now she needed to do something about it.

Chapter 12: The Fork

The first contract came from Covenant Health, a hospital network in the Pacific Northwest with twelve facilities and a combined budget of $3.4 billion. Their chief medical officer, a surgeon named Dr. James Albright, had watched Ava's testimony and called her office — her real office, the one in Ballard — the following Monday.

"I don't understand software," he said, with the particular honesty of a man who was used to being the smartest person in most rooms and was comfortable acknowledging the rooms where he wasn't. "But I understand systems. I've spent thirty years operating inside one. And what you described — the monoculture problem — that's the same thing we worry about with antibiotic resistance. One vulnerability, everywhere."

Ava's team — she still hadn't named it, and the absence of a name had become a kind of identity in itself — began the audit the following week. They worked in pairs: Ava and Priya on the patient-monitoring systems, Tomás and Kwame on the electronic health records platform, Gus and Mei-Lin on the financial and billing infrastructure.

Henrik served as what he called "the old man who reads everything twice," a roving reviewer who checked the others' work and contributed a historical perspective that grounded their analysis.

The work was slow. Deliberately slow. They read code the way a radiologist reads a scan — methodically, section by section, looking not for what was wrong but for what might become wrong. They documented their findings in a format that non-technical executives could understand: each finding was accompanied by a plain-language explanation, a risk assessment, and a recommendation.

After Covenant Health, a European utility company called. Then a bank in Singapore. Then a second bank, in Frankfurt. Then the US Department of Energy, which wanted an audit of three nuclear-plant monitoring systems that had been rebuilt by Architect the previous year.

Within six months, the team had more work than it could handle.

Ava hired. This was harder than she expected. The pool of people who could genuinely read AI-generated code was small and getting smaller — the skill atrophied quickly

without practice, like a language you stop speaking. She placed ads on the Ex-Devs forum, on LinkedIn, and through word of mouth. Most applicants were former programmers who could still write code but had lost the particular skill of *reading* it — the patience, the attention to structure, the ability to see assumptions that the code itself didn't know it was making.

Two hires stood out.

The first was a woman named Noor Bhat, twenty-two, who had never learned to programme. She had a degree in philosophy from Reed College and had been working as a restaurant manager when she applied. Her cover letter was three sentences long: "I read your Senate testimony. I don't know code. But I know how to read systems — I've been running a restaurant with forty employees and a hundred things that can go wrong every night."

Ava almost dismissed the application. Then she read it again. Then she invited Noor to the office and gave her a test: a printed page of Architect-generated code with a subtle architectural flaw hidden in it. She didn't ask Noor to find the flaw. She asked Noor to describe what the code was doing, as best she could, the way you'd describe a process you were observing for the first time.

Noor sat with the printout for forty minutes. She couldn't read the syntax. She couldn't identify variables or functions. But she traced the flow of the code from top to bottom, identified the points where decisions were being made, and produced a diagram — hand-drawn, on a napkin — that accurately mapped the code's logic.

"I can't read the words," she said. "But I can see the shape."

Ava hired her. Noor became the team's most unconventional member and, within three months, one of its most effective. She brought a perspective that none of the programmers had: she saw code as a *decision-making process*, not as a set of instructions. She asked questions that the engineers would never have thought to ask: "Why does this decision happen here and not earlier?" "What happens to the information that this branch discards?" "Is there a reason the system never reconsiders this choice?"

The second notable hire was someone Ava already knew: Ravi Patel.

He showed up at the Ballard office on a Tuesday afternoon, unannounced, looking slightly sheepish.

"Before you say anything," he said, "I still think prompt engineering is valuable. I still think the transition was, on balance, positive. I still think Architect builds better code than humans."

"That's a lot of qualifications."

"I've been thinking about what you said. About the fifteen patients per year. About the tests being written by the same AI."

"And?"

He sat down. He looked, for the first time since she'd known him, uncertain. Ravi had always been a man of forward motion — confident, adaptable, optimistic to the point of occasionally being naive. Uncertainty didn't suit him, but it made him more real.

"I prompted Architect to build a healthcare system last month. Routine contract. I specified the drug-interaction checker, included the standard requirements, everything by the book. And then I looked at the output. Really looked. For the first time in a year, I read the code instead of just reviewing the test results."

"What did you see?"

"The same pairwise model you flagged. Exactly the same. And I thought: I've been building these systems for a year. How many of them have this limitation? I don't know. Because I never looked."

"You want to learn to read code again."

"I want to learn to read code the way you do. Not as a programmer. As a — what did Sarah Kim call you? A systems reader."

"I didn't coin the term."

"It's a good term."

Ava considered him. Ravi was smart, technically fluent, and motivated. He was also, she suspected, the future: the bridge between the old world of programmers and the new world of AI orchestrators. If people like Ravi could learn to read as well as prompt, the gap might begin to close.

"Welcome to the team," she said. "The pay is worse than consulting."

"I know. I checked."

Marcus Adler called in February. Not his assistant. Not his communications team. Marcus himself, calling from what sounded like a car.

"I've been doing some thinking," he said.

"You mentioned."

"We're building a resilience layer into Architect. A system that evaluates the architectural assumptions in its own output and flags potential brittleness. It's modelled, in part, on the findings from your Covenant Health audit."

"That's a good start."

"It's not enough. I know it's not enough. You need human reviewers for critical systems. We need an independent audit capability."

"Are you asking me to work for you?"

"I'm asking you to work *with* me. An independent verification programme. Your team audits Architect-built systems in critical sectors. We provide access and funding. You retain full independence. No editorial control, no NDA on your findings."

"That's a significant offer."

"It's a significant problem."

"Why now?"

There was a pause. Through the phone, Ava could hear traffic. A horn. The whirr of an electric motor.

"Because I ran your team's simulation last week. The cascade-failure scenario. I ran it on our internal systems — the ones that run Synthesis itself. Our own infrastructure."

"And?"

"And the results were sobering."

"How sobering?"

"Sobering enough that I'm calling you from a car on a Tuesday afternoon instead of waiting for my team to schedule a meeting."

Ava looked out the window of the Ballard office. Priya and Noor were at the whiteboard, diagramming a power-grid architecture for the DOE audit. Henrik was reading a printout, pencil in hand, muttering about memory-allocation patterns. Gus was on a call with the Frankfurt bank.

"Send me the proposal," she said. "I'll review it with my team."

"Thank you, Ava."

"Don't thank me yet. My team is going to be very thorough."

"I'm counting on it."

June Watanabe sent a photograph. It arrived by text — one of her biweekly Tuesday communications — and showed a completed dining table, eight seats, in American black walnut. The grain was extraordinary: cathedral patterns in the heartwood, sapwood edges left as accents rather than trimmed away. The joinery was invisible, which meant it was perfect.

The message read: *Some things take time. This one took nine months.*

Ava set the photo as her phone's wallpaper.

That evening, she sat in the Ballard office alone after the team had gone. She was writing a curriculum — a structured programme for teaching non-programmers to

read AI-generated systems. Not to write code. Not to prompt AI. To *read* — to look at the output of a machine and understand the decisions embedded in it.

She thought about June's table. Nine months of hand tools, patience, and understanding. A table that a machine could have built in minutes but that no machine could have *understood* in nine months.

She thought about Leo, building his debate app without writing a line of code. She thought about Noor, reading code she couldn't parse by tracing the shape of its decisions. She thought about Ravi, coming back to reading after a year of prompting, like a musician who'd been singing karaoke and wanted to hear the orchestra again.

She thought about Marcus Adler, calling from a car because his own simulation had scared him.

She saved the curriculum document and closed her laptop. For the first time in a very long time — perhaps for the first time since the morning she'd compiled her last function at Meridian — Ava Chen felt that the work she was doing mattered. Not in the abstract, not in the way that all work arguably matters, but in the specific, tangible way that a bridge matters when the storm is coming.

The storm hadn't arrived yet. But the bridge was being built.

Chapter 13: Compile Again

The classroom was in Denny Hall, the oldest building on the University of Washington campus. Wooden floors that creaked, tall windows letting in the grey Seattle light, and the smell of floor wax and paper and the accumulated warmth of a century of thinking. Ava had chosen it because she wanted her students surrounded by something old.

The course was called "Reading Machines: Systems Literacy for a Post-Programming World." It was listed in Interdisciplinary Studies, because it didn't belong to any single department. Something new — or something very old, reframed.

Twenty-eight students. The largest enrolment the department had ever seen for an elective. They came from everywhere: three medical students, two urban planners, four civil engineers, a logistics manager from Amazon, two lawyers specialising in technology policy, an epidemiologist, a journalist, a chef who managed a

restaurant chain's supply systems, five undergraduates who weren't sure what they wanted to be yet, and three retired programmers who audited the course without credit because they wanted to be in a room where people still talked about code.

Ava stood at the front of the classroom on the first day of the spring quarter and looked at them. They looked back. Some were nervous. Some were curious. One — the chef — was eating a scone.

"This course will not teach you to write code," she said. "There are excellent programmes for that. This course is about something different. This course is about reading."

She pulled up a slide. A page of Architect-generated code — the drug-interaction module from Cascadia Regional, now famous from the Senate hearing.

"This is a system that runs in a hospital. It checks whether the medications you've been prescribed will interact dangerously. It was built by an AI. It works, almost always. And it has a flaw — a structural assumption about how drugs interact — that no amount of testing will detect, because the tests share the same assumption."

She changed the slide. A diagram — Noor's diagram, drawn on a napkin eighteen months ago, the one that had mapped the code's logic without understanding its syntax.

"This diagram was drawn by a member of my team who has never written a line of code. She read the system the way you might read a process you've never seen before — by tracing the flow of decisions. Where does information enter? Where does it branch? Where is information discarded? What assumptions does the system make about what's possible and what isn't?"

"That's what this course teaches. To look at the systems that run your hospital, your city, your bank, and understand the decisions embedded in them. Because if you can't read them, you can't question them. And if you can't question them, you can't change them."

The course met twice a week. Tuesdays were lectures. Thursdays were labs — printouts, whiteboards, group analysis of real AI-generated systems. Ava brought in guest speakers: Priya on security assumptions, Tomás on architectural patterns, Henrik on the history of software

failures. Ravi came once, to talk about the view from the other side — what prompt engineers could see and what they couldn't.

The students surprised her.

Dr. Amira Hassan, one of the medical students, read the drug-interaction code and immediately grasped the tertiary-interaction problem. She understood pharmacology in a way that neither Ava nor Architect did. She didn't need to read the syntax. She understood the *domain* — the gap between what the code modelled and what the body actually did.

Kenji Nakamura, one of the urban planners, read a traffic-management system and identified an assumption that no one on Ava's team had caught: the system modelled pedestrian traffic using vehicle-traffic patterns, because vehicle data was abundant in the training set and pedestrian data was sparse. The result was a system that optimised for cars and treated people on foot as noise.

"That's not a bug," Kenji said. "That's a worldview. The system believes that streets are for cars."

"Correct," Ava said. "And now that you can see it, what do you do about it?"

"You change the spec. You tell the AI to model pedestrians as a primary variable, not a secondary one."

"And if you can't read the system, can you do that?"

"No. You'd never know the assumption existed."

"That," Ava said, "is why you're here."

A question came near the end of the quarter. It was asked by Lily Okonkwo, one of the undergraduates — a second-year student with a double major in philosophy and environmental science who had been, throughout the course, the most consistently challenging questioner in the room.

"Professor Chen," she said, "I understand the argument. Systems have assumptions. Assumptions can be wrong. If nobody reads the systems, nobody catches the wrong assumptions. But here's my question: if the AI writes better code than humans ever did — and the data says it does — why does it matter if we understand it? We don't understand how the sun works, not really, but we still use solar panels. We don't understand consciousness, but we still think. Why is *software* the thing we need to understand?"

The room waited. It was the question Ava had been expecting since the first day — the question that, if she was honest, she didn't have a perfect answer to.

"You're right that we use many things we don't fully understand," she said. "But there's a distinction. The sun exists independently of us. We observe it. We model it. If our model is wrong, the sun doesn't change — our predictions do, and we adjust. Software is different. Software is a *decision* that we've delegated to a machine. Every line of code is a choice about how something should work. When we don't understand those choices, we're not failing to understand nature. We're failing to understand our own decisions."

She paused.

"Put it another way. If I give you a book to read and you can't read the language it's written in, you might say: 'It doesn't matter. Someone translated it for me, and the translation is excellent.' And maybe it is. But a translation is an interpretation. Something is always lost. Something is always added. And if you can never compare the translation to the original, you can never know what was lost or added. You just have to trust."

"And trust is the thing you think we shouldn't do?"

"Trust is the thing I think we should earn. Not assume."

Lily nodded. She didn't look convinced, and Ava respected that. The question was genuine, and the answer was incomplete. It would always be incomplete. That was the nature of the problem.

The call from Leo came on a Thursday evening in May. Ava was in her office at the university, grading lab assignments — an activity she found deeply satisfying, because her students' work was genuinely interesting, and because she could see them learning to see.

"Mum." Leo's voice was excited in a way she hadn't heard since he was twelve and had built his first Lego Technic set. "I need your help."

"What's going on?"

"Rebuttal. The debate app. It's gotten big. Like, really big. Forty thousand users. Two schools in Japan adopted it. The debate team at Oxford is using it."

"That's wonderful, Leo."

"Yeah, it is, except there's a problem. The real-time system is crashing under load. Users get disconnected in the middle of practice sessions. I've asked Architect to fix it, and it keeps generating solutions that work for a while and then fail again."

Ava smiled. She couldn't help it. "What kind of solutions?"

"It keeps adding more server instances. More memory. More bandwidth. But the problem keeps coming back."

"Because the problem isn't resources. It's architecture."

"That's what I thought. But I can't—" He stopped. She could hear him wrestling with the words, the particular frustration of someone who understood that there was something he didn't understand. "I can't see *why*. I know the system is failing. Architect knows the system is failing. But neither of us can figure out the root cause."

"Do you remember what I told you about the session-management module? A year and a half ago?"

"The in-memory sessions thing. You said it would be a problem if the app scaled."

"It's a problem now."

"How do I fix it?"

"You need to understand the architecture. Not just what it does — how it's structured. The relationships between the components. The assumptions baked into the design."

"Can you teach me?"

The question landed on Ava like sunlight. Leo, who had never needed to understand the code, was asking to understand the code. Not because the old world demanded it. Because the new world required it.

"I'll send you some reading," she said.

"Reading? Like a textbook?"

"Like the first four modules of my course. Start with the systems-literacy framework. Then read the section on architectural assumptions. Then call me back and we'll look at your codebase together."

"Together?"

"Together. I'll read. You'll learn to read. We'll figure it out."

"Thanks, Mum."

"Leo?"

"Yeah?"

"I'm proud of you. Not because you're asking for help. Because you're asking the right question."

She hung up. She sat in her office and looked at the stack of lab assignments on her desk. Outside the window, the campus was green with late spring, the cherry trees dropping petals on the paths where students walked with their heads down, looking at phones, talking to AIs, building things they would one day need to understand.

She reached into her desk drawer and took out the bookmark June had given her. Cherry wood, hand-carved, a leaf in shallow relief. She'd been using it in the textbook she was writing — *Reading Machines: A Primer for the Post-Programming World*. The book was half-finished. It would take time.

She placed the bookmark between the pages she'd been working on and closed the book.

That night, at home, she opened her laptop one more time. Not to write code. Not to prompt an AI. To read.

On her screen: a page of Architect-generated code. A logistics system. Millions of lines she would never read all of, but this page — these eighty lines — she could read.

She could see the decisions embedded in them, the assumptions that would hold until they didn't.

She read the way June worked wood: slowly, attentively, with hands that remembered.

The cursor blinked. Green on black. The same green, the same black, as the terminal she'd closed at Meridian two years ago. But reading was different from writing. Writing was creation. Reading was comprehension. And in a world where the machines did the writing, comprehension was the thing that mattered most.

Outside, Seattle settled into its evening. The rain had stopped. The lights of the city reflected in the wet streets, each one connected to a system, each system built by an intelligence that was brilliant and tireless and incapable of doubt. Somewhere in those systems, assumptions were hardening into certainties. Somewhere, a variance was drifting. Somewhere, a safety margin was being optimised away.

But in a classroom in Denny Hall, twenty-eight people were learning to read. And in an office in Ballard, seven people were still reading. And in a dorm room at MIT, a

boy who had never written a line of code was opening a textbook his mother had written and seeing, for the first time, the shape of what was underneath.

The cursor blinked.

Ava read on.

About the Author

Kael Eriksson writes speculative fiction set five minutes into the future. His stories take real technology trends and ask: what happens to ordinary people when this goes wrong? A former software consultant based in Stockholm, he now writes full-time from a converted boathouse on the Baltic coast. *The Last Programmer* is his first novel for Kelford Press.

Also by Kelford Press

The Attention Economy Is Dying: And What Comes After *Eleanor Whitfield* A sharp, deeply reported investigation into the collapse of the ad-funded internet and what replaces it.

Daily Papers Award-winning journalism across Technology, Business, Science, Culture, and World News — published every day.

Weekly Digest The essential stories, curated and contextualised — delivered every Friday.

Monthly Letters A personal editorial letter from the desk of the Editor-in-Chief — the first of every month.

Become a Member

Kelford Press is an independent publisher. We are funded by our readers, not by advertisers.

Reader — Free 5 articles per month, digest previews

Subscriber — $4.99/month Unlimited papers, full weekly digest, monthly letters

Patron — $9.99/month Everything above + digital books + audibles + ad-free reading

Fellow — $79.99/year Patron benefits + quarterly signed book + exclusive events

Join at **kelfordpress.com/subscribe**

Kelford Press · Where Words Find Their Home · Est. 2006