

**MAX PARADOX**

# **ABSURDITY — OF BEING A — PROGRAMMER**

**Deadlines, Production Bugs,  
Impostor Syndrome, AI Panic —  
and the Quiet Pride No One Sees**



|                                                                         |     |
|-------------------------------------------------------------------------|-----|
| INTRO.....                                                              | 3   |
| Chapter 1 – The Myth of the Logical Profession .....                    | 7   |
| Chapter 2 – It Worked on My Machine .....                               | 13  |
| Chapter 3 – The Meeting That Could Have Been an Email ...               | 19  |
| Chapter 4 – The Senior Developer Who Knows Too Much ..                  | 25  |
| Chapter 5 – The Framework That Will Save Everything .....               | 31  |
| Chapter 6 – Technical Debt Is Just Emotional Debt with<br>Syntax .....  | 37  |
| Chapter 7 – The Bug That Only Happens in Production at 2<br>A.M.....    | 43  |
| Chapter 8 – Estimation Is Performance Art.....                          | 49  |
| Chapter 9 – Impostor Syndrome with a Keyboard .....                     | 56  |
| Chapter 10 – The Holy War of Tabs vs Spaces.....                        | 63  |
| Chapter 11 – The Code Review That Feels Personal .....                  | 69  |
| Chapter 12 – The Sprint That Never Ends.....                            | 75  |
| Chapter 13 – Documentation Is a Love Letter No One Reads<br>.....       | 82  |
| Chapter 14 – The Stack Overflow Tab That Never Closes ...               | 88  |
| Chapter 15 – The Feature Creep That Looks So Innocent ....              | 95  |
| Chapter 16 – The Build That Fails for No Reason.....                    | 101 |
| Chapter 17 – The Deadline That Was Always Fiction.....                  | 108 |
| Chapter 18 – The Side Project That Will Change Everything               | 115 |
| Chapter 19 – The Refactor That Broke Everything.....                    | 122 |
| Chapter 20 – The Manager Who Thinks It's Just One Line of<br>Code ..... | 129 |

|                                                                           |     |
|---------------------------------------------------------------------------|-----|
| Chapter 21 – The AI That Will Replace You (Any Minute Now)                | 136 |
| Chapter 22 – The Environment Variable That Ruined Everything              | 143 |
| Chapter 23 – The Legacy System No One Dares to Touch                      | 150 |
| Chapter 24 – The Perfect Architecture That Existed Only in the Slide Deck | 157 |
| Chapter 25 – The Merge Conflict That Tests Your Character                 | 163 |
| Chapter 26 – The Productivity System That Will Finally Fix You            | 170 |
| Chapter 27 – The Salary That Sounds Higher Than It Feels                  | 176 |
| Chapter 28 – The Burnout That Arrives Quietly                             | 183 |
| Chapter 29 – The Remote Work Paradox                                      | 190 |
| Chapter 30 – The Endless Learning Curve                                   | 197 |
| Chapter 31 – The Production Incident Postmortem That Changes Nothing      | 203 |
| Chapter 32 – The Interview Where You Forget How to Reverse a String       | 210 |
| Chapter 33 – The Comment That Says "This Is Temporary"                    | 217 |
| Chapter 34 – The Day You Realize You're the Senior Now                    | 223 |
| Chapter 35 – The Quiet Satisfaction No One Sees                           | 230 |

# INTRO

Being a programmer sounds impressive right up until you say it out loud in a normal conversation.

People nod.

People smile politely.

People immediately assume you either make millions or fix printers.

Sometimes both.

Programming is one of those professions that exists mostly in other people's imaginations. In those imaginations, you sit in a dark room filled with glowing screens, typing aggressively while green text scrolls down like a movie hack. You solve problems in seconds. You drink coffee like a personality trait. You wear a hoodie that signals intelligence instead of laundry neglect.

None of this is true.

All of this is somehow also true.

Being a programmer is not about writing code. It is about negotiating with reality using a keyboard. It is about convincing machines to do extremely simple things in the most complicated way possible. It is about spending eight hours explaining to a computer what you meant, and another eight hours explaining to humans why it took that long.

Programming is the only job where you can break everything without touching anything.

You don't move desks.  
You don't lift boxes.  
You don't even leave your chair.

Yet somehow, production is down, sales are frozen, and someone in management is asking if this was "a simple change."

It never is.

The absurdity begins early. Usually with the belief that programming is logical. Clean. Rational. A profession governed by rules, certainty, and if-then statements. This belief lasts until your first real project. Or your second. Or about fourteen minutes into your career.

After that, you learn the truth.

- Logic is optional.
- Consistency is aspirational.
- Documentation is a myth passed down orally and immediately forgotten.

You learn that code does not age like wine. It ages like milk left in a warm car. You learn that the thing you wrote six months ago was clearly written by a different, less intelligent person who should no longer be trusted.

That person was you.

You also learn that programming is never done. It is merely abandoned in a temporarily functioning state. Features pile up. Bugs mutate. Workarounds become core architecture. The system keeps running mostly because everyone involved is afraid to touch it.

Especially you.

There is a quiet exhaustion baked into this profession. Not the dramatic kind. Not burnout with flames and breakdowns. Just a low, constant tiredness from thinking too much about things that should not require thinking.

- Why does this work only on Tuesdays.
- Why does changing one line break something unrelated.
- Why does it work in production but not on your machine.

You stop asking “how” and start asking “how long until this fails again.”

And yet, you stay.

You stay because somewhere between the bugs, the meetings, and the polite panic, there is a strange satisfaction. A moment when something finally works and no one knows why. A brief, fragile silence where the system behaves, users don’t complain, and you are allowed to exist without explaining yourself.

It never lasts.

Soon enough, someone asks for a “small tweak.” Someone else adds urgency. Someone mentions scalability. Someone forwards an email that ends with “should be quick.”

You take a deep breath.

You open the code.

You remember why you drink coffee.

This book is not here to teach you programming.  
It assumes you already suffer from it.

It is here to observe.

To point.

To linger uncomfortably on the parts everyone pretends are normal.

Because being a programmer is absurd.

And pretending it isn't takes more energy than fixing the bug.

# Chapter 1 – The Myth of the Logical Profession

The first lie you were told about programming was that it is logical.

Not creative.

Not emotional.

Not chaotic.

Logical.

You were promised a world of precision. A universe where inputs lead to predictable outputs. A professional environment governed by reason, clarity, and clean structure.

You believed it.

You imagined yourself living inside a perfectly structured mind palace made of brackets and semicolons. A world where everything makes sense because you made it make sense.

Then you joined your first real project.

And suddenly logic became... negotiable.

Programming is logical in the same way that cooking is precise. In theory, recipes are clear. In reality, someone always eyeballs the salt and forgets the oven was already preheated to chaos.

Codebases are not temples of order. They are archaeological sites.

- There are ancient ruins no one dares to touch.
- There are mysterious layers built on top of older mistakes.
- There are comments written in languages no one on the team speaks anymore.

Somewhere inside this structure, you are told, there is logic.

You just have to find it.

The myth of the logical profession collapses slowly. It doesn't explode. It erodes. One meeting at a time.

In theory, requirements are clear.

In practice, requirements are feelings.

"We want it to be more dynamic."

"Can it feel faster?"

"It should be intuitive."

You nod like you understand. You do not understand.

You translate vague human emotion into binary decisions. You reduce "vibes" into conditional statements. You try to convert "make it pop" into something that compiles.

This is where the logic begins to sweat.

- The backend logic is clean.
- The frontend logic is complicated.
- The business logic is imaginary.

No one tells you that programming is less about writing code and more about interpreting ambiguity. You are not building systems. You are decoding unclear intentions from people who believe software is magic.

They assume the machine understands context.

The machine does not understand context.

The machine understands exactly what you tell it. Nothing more. Nothing less. And often not even that.

This is where the absurdity sharpens.

You write something perfectly logical. Elegant. Structured. Even beautiful. Then someone uses it in a way you never predicted.

Because humans are not logical.

They click the wrong button. They refresh at the wrong moment. They enter their birth year as their password and then complain about security.

And somehow this becomes your problem.

The myth says programmers live in rational isolation.

The reality is different.

You are constantly negotiating between three incompatible worlds:

- What the machine requires.
- What the business demands.
- What the user does at 2:13 a.m. on a slow connection.

These worlds do not align.

The machine wants precision.

The business wants speed.

The user wants magic.

You are the translator.

And translators are always blamed.

Then there is "clean code." The holy grail. The promise that if you follow certain principles, your world will remain orderly and pure.

You try.

You name variables responsibly. You separate concerns. You refactor. You test. You even comment your intentions.

Six months later, someone adds a "temporary fix."

Temporary fixes have an average lifespan longer than most startups.

One quick workaround becomes a dependency. That dependency becomes architecture. That architecture becomes sacred because "it works, so don't touch it."

Logic did not fail. It was slowly negotiated into something survivable.