

MAX PARADOX THE ABSURDITY OF BEING A PROGRAMMER



*A brutally honest satire about code, burnout,
meetings, bugs, and the quiet
madness behind the screen.*

INTRO.....	3
Chapter 1 - Everyone Thinks You Fix Printers	7
Chapter 2 - It Works on My Machine.....	14
Chapter 3 - Debugging Is Just Structured Panic	22
Chapter 4 - You Don't Actually Know Any Programming Languages	31
Chapter 5 - Meetings About Code, Without Code	38
Chapter 6 - You Are Always On Call, Even When You Aren't	46
Chapter 7 - Legacy Code Is Just Code That Survived You ..	55
Chapter 8 - Your Job Is Explaining Why Things Take Time ..	63
Chapter 9 - You Google for a Living	71
Chapter 10 - Deadlines Are Negotiations, Not Facts	78
Chapter 11 - Everyone Thinks Automation Means Less Work	85
Chapter 12 - Impostor Syndrome Never Leaves, It Just Gets Quieter	93
Chapter 13 - Burnout Doesn't Arrive Loudly	99
Chapter 14 - Seniority Means Saying No Politely	106
Chapter 15 - Nobody Thanks You for What Didn't Break	112
Chapter 16 - The Code Is Never "Done"	118
Chapter 17 - You Are Not Paid to Type.....	126
Chapter 18 - Every System Is Held Together by Assumptions	133
Chapter 19 - Context Switching Is the Real Productivity Killer	140

Chapter 20 - You Eventually Stop Arguing with Reality	148
Chapter 21 - You Learn to Read Between the Requirements	153
Chapter 22 - You Become the Person Who Says "It Depends"	159
Chapter 23 - You Stop Trying to Be the Smartest Person in the Room.....	165
Chapter 24 - The Tooling Will Change, the Problems Won't	172
Chapter 25 - Experience Is Knowing Where the Bodies Are Buried.....	178
Chapter 26 - You Start Designing for the Person After You	184
Chapter 27 - At Some Point, You Stop Taking Bugs Personally	191
Chapter 28 - You Realize the Job Is Mostly About Trust ...	197
Chapter 29 - You Stop Measuring Yourself by Output.....	204
Chapter 30 - You Accept That You'll Never Catch Up	210
Chapter 31 - You Stop Looking for Permission.....	216
Chapter 32 - You Learn That Simplicity Is a Discipline.....	222
Chapter 33 - You Realize Teaching Is Part of the Job	229
Chapter 34 - You Stop Romanticizing the Job	236
Chapter 35 - In the End, You're Still Just Solving Problems	241

INTRO

Every profession claims it is misunderstood.
Programming is the only one that proves it daily.

From the outside, being an IT specialist looks like a controlled, almost meditative experience. A person sits quietly in front of a screen. Coffee appears regularly. Nothing explodes. Nobody screams. Money arrives anyway. It looks like the closest thing modern civilization has to cheating the system.

From the inside, it feels like living in a house where the walls rearrange themselves at night and then blame you for not remembering yesterday's layout.

You were told this was about logic.
You discovered it was about vibes.

You were promised clarity.
You learned to survive ambiguity instead.

You entered a field where precision is worshipped, yet everything works "for reasons we'll investigate later."

- The code compiles, but nobody knows why.
- The code fails, and it is definitely your fault.
- The same code did both, five minutes apart.

This is a book about that space between expectation and reality.

Not the heroic version of programming shown in conference talks.

Not the LinkedIn version with pastel diagrams and confident smiles.

But the quiet, absurd middle, where most developers actually live.

Here, productivity is measured in coffee refills.

Progress is defined as "nothing got worse."

And success often means the system didn't notice you today.

Programming is not about writing code.

It is about negotiating with systems that remember everything except why they exist.

You inherit projects with names like "final_v3_fixed_REAL."

You open files last modified by someone who has since changed careers, countries, or personalities.

You read comments that confidently explain behavior that no longer happens.

- Someone once knew what this did.
- That person no longer answers messages.
- You are now that person.

There is a special kind of exhaustion that comes from solving problems that technically shouldn't exist.

Bugs that vanish when observed.

Errors that politely wait until Friday evening.

Features that work perfectly, except in production, for emotional reasons.

You learn new languages constantly.

Not because you want to.

But because the industry wakes up every year and decides yesterday's tools are now "legacy."

- You just mastered something.
- It is already a bad career choice.
- There is a conference about its replacement.

The job demands total focus while actively punishing confidence.

Every solution feels temporary.

Every certainty comes with a silent expiration date.

You are expected to be logical, fast, creative, precise, flexible, and calm.

Preferably before lunch.

And yet, despite everything, people keep coming back.

Not because it's easy.

But because somewhere between frustration and flow, there are moments when things align.

A function behaves.

A system responds.

A problem dissolves instead of multiplying.

Those moments are brief.

They do not last.

They are enough.

This book does not explain how to become a better programmer.

It will not teach best practices.

It will not optimize your workflow or fix your posture.

It will simply sit next to you, nod occasionally, and say:

- Yes, this is strange.
- No, you are not imagining it.
- Everyone else is pretending harder.

If you have ever stared at a screen waiting for meaning to appear.

If you have ever whispered "why" to a machine that refused to answer.

If you have ever fixed something by changing nothing.

You are in the right place.

Chapter 1 - Everyone Thinks You Fix Printers

At some point, someone decided that “working with computers” is a single skill.

It didn't matter whether you design distributed systems, optimize databases, or quietly prevent civilization from collapsing at 3 a.m.

In the public imagination, you became The Computer Person.

Which means one thing.

You fix printers now.

It starts innocently.

A family gathering.

A friendly tone.

A sentence that sounds harmless.

“So... you work in IT, right?”

You already know what's coming.

Your body knows it before your brain does.

Your shoulders tense.

Your soul attempts to disconnect.

- This is not a conversation.
- This is a support ticket without a ticket number.
- Payment will be gratitude and confusion.

Nobody ever asks what you actually do.

They ask what their device is doing wrong.

Their laptop is "slow."
Their phone is "acting weird."
Their Wi-Fi "just doesn't feel the same anymore."

Details will not be provided.

- It was working yesterday.
- Nothing was changed.
- Except everything.

You try to explain that software development and hardware troubleshooting are different disciplines.

They nod politely.

Then they hand you a cable.

In their mind, computers are magic boxes.

You are a wizard.

Wizards fix all magic boxes.

It doesn't help that sometimes you actually can fix it.

Not because it's your job.

But because the bar is low and the problem is usually fear.

You restart the device.

You wait.

It works.

The room goes quiet.

- You are now dangerous.
- Expectations have been updated.
- This will happen again.

From that moment on, your identity shifts.
You are no longer a person who writes code.
You are technical support with a personality.

Someone's email won't send.
Someone's TV remote stopped responding emotionally.
Someone forgot their password and wants you to "hack it."

They say "hack" the way people say "microwave."

- They assume it's quick.
- They assume it's legal.
- They assume you enjoy it.

You explain that hacking is not a thing you casually do for relatives.
They laugh, because they think you're joking.

Movies did this.
TV shows did this.
Some guy once fixed a printer and ruined everything for the rest of you.

On screen, the computer expert types fast.
Green text scrolls.
Problems collapse in seconds.

In reality, most fixes involve waiting.
Waiting for updates.
Waiting for logs.
Waiting for someone else to admit they touched something.

- The most powerful command is "restart."
- The second most powerful is "wait."
- Neither looks impressive.

Your job becomes a paradox.

If things work, nobody knows what you do.
If things break, everyone assumes it's your fault.

You are judged by outcomes you cannot fully control,
in systems you did not design,
using requirements written by people who fear computers
but trust them with everything.

And still, people believe you have universal answers.

They ask you which laptop to buy.
You ask what they need it for.
They say "normal stuff."

- Normal does not exist.
- Normal is a lie people tell before buying the wrong thing.
- You will be blamed later.

You try to explain trade-offs.
They want a simple recommendation.

Fast.
Cheap.
Light.
Powerful.
Reliable.

All at once.

You suggest something reasonable.
They buy something else.
It breaks.