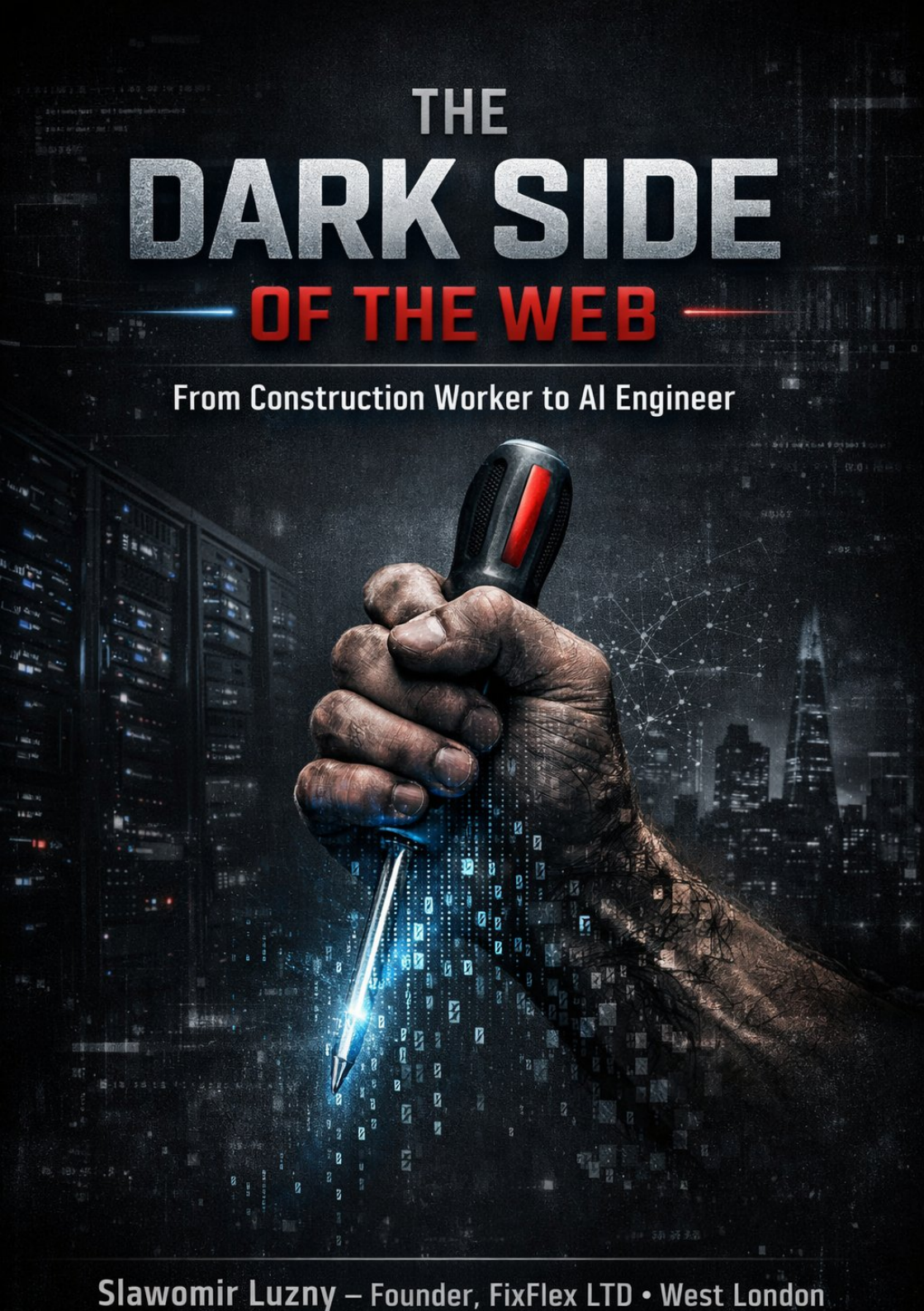
THE DARK SIDE OF THE WEB

From Construction Worker to AI Engineer



Slawomir Luzny – Founder, FixFlex LTD • West London

The Dark Side of the Web

How I went from construction worker to AI engineer — fighting corporations, bots, and my own body

Slawomir Luzny — Founder, FixFlex LTD, West London, 2026

Table of Contents

Chapter 1: A Screwdriver in the Cloud World

If you open today's programming textbooks or read the tech blogs of corporate "evangelists," you'll always hear the same tune. They'll tell you that to build advanced systems you have to finish university, memorize the system specification, and spend five years mindlessly grinding out algorithms.

That's nonsense.

My adventure with technology didn't start in a sterile office in Silicon Valley, or even in a design office in West London. It started much earlier, back when the peak of home technology was the Commodore C64 and code was written in Basic. I still remember that childish, pure satisfaction when, line by line, I typed in instructions that made a balloon appear on the CRT screen and let me steer its flight with the cursor. I didn't understand processor architecture back then. A friend of mine, an absolute genius of those days, could write a whole Minesweeper in assembly on the C64. Next to him I was a pawn. But that's exactly when I caught the bug. I understood that a computer isn't just a machine for playing games — it's a tool that does exactly what you tell it. As long as you can think logically.

Life turned out differently than I planned, though. I didn't become a programmer right away. I had to do all sorts of things, earn my bread, work with my hands. But that technical curiosity never died in me. It got to the point where, before I even owned my own desktop computer, I was buying and devouring thick books about Windows Office and computer magazines. I studied the descriptions, the diagrams, learned dry — where you click and what follows from what. When I finally went into the army and they sat me in front of a computer, people rubbed their eyes in disbelief. I moved around the system like an old hand, even though I was physically seeing the hardware for the first time in my life. I simply had it in my head.

Later I earned my IT technician certificate. I assembled computers, overclocked processors, dug through directory structures, learned DOS commands and database relations in Access 2000, porting them into C++ structures. And it was exactly that raw, purely practical school of life that gave me something you can't buy at any university: a technical "nose." An understanding of the physical nature of hardware.

Years later, when I was working in the fields in Spain, I got talking with a guy. He complained that the Spanish IT people had thrown up their hands over his machine. Before me, five "professional IT technicians" had seen that computer. One of them had just installed a fresh

Windows, ran advanced benchmarks, tested software, took the money — and the computer still shut itself off after a few minutes.

I went over to him. I looked at their benchmarks and said:

"Alright, turn that off, we're not installing anything. Bring me a screwdriver."

The guy and his father looked at me like I was crazy. How's that? An IT guy with no rescue USB stick, no clever expressions, just a screwdriver in hand?

I unscrewed the case, looked inside, and it hit me right away. One small clip on the processor cooler wasn't fastened. The fan wasn't making contact, the processor was hitting a huge temperature, and the system was cutting power so it wouldn't burn out. That was the whole "magic." I pressed the clip down, screwed the case back together, fired up the benchmark. Everything ran like it should. Five theorists had failed because they were looking for problems in the cloud and the code, while the problem lay in a piece of plastic and physical overheating.

Today's internet and web environments suffer from exactly the same disease. They've become an inflated, virtual cloud where people let themselves be manipulated by big corporations and complicated tools,

forgetting about simple, common-sense solutions. This book is about tearing the mask off that digital circus.

Chapter 2: War on Bloatware and the Power of Clean APIs

For the first year of building my first serious project, I lived under a painful illusion. I thought that since I was paying serious money for something, I was getting quality. That was the most expensive mistake of my life.

For two years I paid a monthly cPanel subscription. Why? Because everyone around me said that's how it's done, that without a graphical panel you can't manage a server. I paid for professional mailboxes on Zoho, bought ready-made scripts. The reality? I got one big pile of bloatware — systems overloaded with thousands of features nobody needs, forever demanding updates, choking memory, and creating security holes. When a failure hit, cPanel just blinked its indicator lights helplessly.

That's when I understood that if I wanted to survive, I had to throw off all that corporate ballast. I stopped fooling around with free half-measures and sluggish ready-mades. I did something traditional programmers look at with horror. I took WinBinder and wrote my own raw file manager. No bells and whistles. Clean upload, download, and a simple verification of the directory structure on the server. It did exactly what I needed and didn't ask permission to install yet another plugin.

But the real war was only coming. I started experimenting with server architecture in a way that defied every textbook rule. I tried to bind into one powerful engine Apache, Nginx, LiteSpeed, Caddy, Varnish, and I even tested Rust. Someone will say: "Why mix so many technologies at once?" Because I was looking for the holy grail of performance. I wanted to create the perfect engine. Of course, the server choked. Ports refused to obey, redirects looped endlessly, and packets got lost somewhere in the network configuration. Technically it had no right to work in that configuration. But every one of those sleepless nights, every fight with a closed port, taught me the map of the server better than any university could.

That phase of experiments led me to the final and ruthless step: completely rejecting PHP, WordPress, and Laravel. I understood that these old technologies are just a mass of hidden technical debt, written by developers who only wanted to squeeze money out of me and disappear. I rewrote my projects — including the classifieds platform 24ad.info — onto the most modern, raw stack: React 19, Vite, Node.js, and Drizzle ORM tied to a MySQL database.

I had to understand from scratch how to think in terms of databases and REST API architecture. To understand how applications talk to the server and how they transmit data. I built my systems on a proven, powerful

foundation: FastAPI, SQLModel, PostgreSQL, and MySQL.

Traditional programmers spend years learning the complicated syntax of these languages, getting lost in brackets and forgetting the business logic. I don't know all those dry C++ or Java commands by heart. And honestly? I don't need to. I understand architecture, I understand the relationships between databases, and I know what the finished system should look like. The rest — under my strict supervision and after my verification — is written by artificial intelligence. I chose a stack so fast and clean that the traditional approach buckles on the curves, while my applications run on servers like lightning.

Chapter 3: The Digital Drug and the Price in Blood

If you think building your own tech business after hours is a beautiful, romantic story of success, you're badly mistaken. It's grind that can slowly, systematically destroy you. And I say that with full responsibility.

I've been in this web world for four, five years now. Five years in which my day split into two completely different worlds. By day — hard, physical work on the construction site. Sweat, material fatigue, carrying loads. At 8 p.m., when a normal person turns on the TV or lies down to rest, my second shift began. The house went quiet, the family went to sleep, and I sat down in front of the monitor. And I disappeared.

Coding with AI, designing databases in PostgreSQL or MySQL, tying everything together through clean, fast APIs — it works like the worst drug. When, after hours of fighting with code, a critical connection suddenly starts working, you get such a hit of dopamine that you don't feel like sleeping anymore. You look at the clock — 1 a.m. You think: "Alright, I'll just wire up these routes, check the database queries, and I'm done." Suddenly something breaks. The server spits out a connection error. And the fever begins. You search, you change prompts, you analyze logs. You wake from the trance

and the clock says 3:45 a.m. In two hours the alarm rings for the construction site.

You can't fall asleep, because tables, relations, and loops are still spinning in your head. You sleep two, three hours. In the morning you get up clouded, with a ringing in your ears, drink another strong coffee, and go put up walls. And so it goes, night after night. For five years.

At some point your body hands you the bill. There are no miracles — a human is not a server with an infinite power supply, even if you have a bulldog's stubbornness in you. I started to feel myself weakening. From one day to the next I had less strength for things I used to do without blinking. Suddenly your heart pounds for no reason, a dull pain throbs in your temples. You go to the doctor, and the numbers on the blood pressure monitor mean one thing: a straight road to a heart attack. Today, every morning, before I even think about servers or projects, I have to swallow pills for high blood pressure. That's my real price for all those sleepless nights. I paid with my own health for every working application.

Iwona's Voice (The Wife's Perspective)

"*Let me tell you what all his 'empire building' looks like from my side. Sławek tells you about servers, about databases, about how AI writes his code. It sounds*

beautiful in books, doesn't it? But do you know how it looked at home?

Midnight. The kids long asleep. I wake for a moment, look at the other half of the bed — empty. I get up, go to the kitchen, and there in the dark only that cursed monitor screen is glowing. And him. Red eyes, staring at some little tables, face tense, as if the fate of the world depended on it. I say quietly, so as not to wake the little ones: 'Sławek, go to sleep, you've got the site in the morning, you'll wear yourself out.' And he doesn't even look at me. He just waves his hand and throws out his classic line: 'Five more minutes, I'll just finish one API, close the database, and I'm coming.' Those 'five minutes' lasted until four in the morning.

In the morning I wake before six. Sławek barely alive, clouded, drinking black coffee and heading out to the site. I walk into the kitchen and my blood boils. The deal was simple: he'd handle the kitchen in the evening and wash the dishes after dinner, so I wouldn't have to start my day with it — when you have to get the kids ready for school, make breakfast, pack the schoolbags, and run around in all that morning chaos. And what? The dishes sit in the sink unwashed. The counter buried, dishes dried on, because the mister engineer forgot about the world, because the server was more important than a promise made to his wife.

I was furious. Not because he wants to achieve something. I was furious because in all of this I felt terribly alone. The whole house, school, the kids, the shopping, the everyday problems — all of it was on my shoulders. And he was physically with us, but in his mind he sat on some cloud. How many times did the kids want to show him something, and he was looking at his phone, checking whether the bots were attacking his site?

The worst was the fear. I watched him grow weaker month by month. A man like an oak, and he'd come back from the site and instead of resting, load himself into that digital trance. When he started having those blood pressure spikes, when the doctor told him to take pills — I broke. I cried and shouted in turns: 'What is all this for? What do I need your programs for, if you're about to have a stroke and the kids won't have a father?!' Those servers of his weren't feeding our children back then, weren't paying the bills. It was just an illusion, a drug that was taking my husband away and the kids' father away. Only when he started managing his time more sensibly did normal air return to our home. Today, when he closes the computer and simply comes to talk with me or finish washing those dishes — I know I got Sławek back, and not a robot."

Chapter 4: Corporate "Marble Games"

When you step into the world of building software with AI, you quickly realize you're not alone in there. You're in the playground of tech giants. Their marketing slogans talk about "democratizing code" and "helping creators." The reality can be harsher: you're the one who's supposed to keep paying.

I went through a lot of tools. I started on some, switched to others, tested a third set. Each had its price — sometimes in money, sometimes in nerves. At first everything looks perfect. You get a raw, brilliant tool running straight from the terminal. No overloaded interface, clean writing, lightning reactions. Plus low costs to start and promotions on cheaper tokens during certain hours. You think: "Finally, something built for developers." You go in deep, you design the architecture, you make your daily workflow dependent on the ecosystem.

And then comes the night when all of it sticks in your throat.

You're sitting over a critical bug. You have to get the routing back on its feet right now, or fix the database queries, because production is up against the wall. And

suddenly you see the token counter melting before your eyes. The daily or weekly limit runs out exactly when you need it most. You're put against the wall — either you make a micropayment and buy a higher limit, or your project freezes halfway. And you know what I felt then? That this is a payment model lifted straight out of mobile games — they pull you in for free, and make you pay the toll exactly when you're most desperate. I'm not claiming someone sits and turns the dials specifically for my late-night session. I'm claiming that's *how I experienced it* after the fifth hour of fighting at 3:45 a.m. — like a loop designed to make me pay more and more.

But with me, that mechanism met the wrong opponent. Instead of panicking and mindlessly pumping in more money, that frustration changed my whole attitude toward work. I said: enough. I won't let myself be pulled into a game where I always lose — with my own health and my own wallet.

I introduced hard discipline. When I sit down in the evening and see the limits approaching the end, I no longer go into that old psycho-phase. I don't throw myself into mindless coding by force until four in the morning. In a split second I make the decision. I close the session. I fire up my internal system — Project Brain — and write a precise note:

```
[session_state]
last_endpoint = fastapi_routing_v2
status = paused
next_action = verify_database_pool
```

At that moment the topic of programming dies for me. I cut the pressure, close the laptop, and go rest, giving my body the regeneration it so badly needs. But that doesn't mean FixFlex LTD stops working. When I lie down to sleep, my own autonomous systems step into the game.

On tools and "vanishing" context

Over these years quite a few AI tools passed through my hands. Each taught me something — and each had its Achilles' heel. On one of them, **Windsurf**, I ran into a problem I call "vanishing." On long sessions the tool lost context — it could fix one bug and break three other things that had been working before. The longer I sat, the more the AI drifted from what I was actually asking for. That's exactly what taught me an iron rule: git became my safety net, and short, controlled sessions became my weapon against drift. I don't blame one tool for the sins of another. Each has its place and its limits — the art is knowing which is for what, and trusting none of them blindly.

Chapter 5: Hypocrisy on the Forums and the Birth of SpamSlap

Before I talk about the machines that work for me, we have to expose one more environment: the community of so-called "traditional programmers." When I started publicly saying that I build advanced, production-running SaaS systems with AI, a bucket of slop was poured on me on the forums. People who cut their teeth memorizing language syntax tried to discredit me. I heard that "a prompter isn't a programmer," that without five years of computer science studies my products are houses of cards about to collapse.

You know what's funniest in all this? Their absolute, pure hypocrisy.

The same people who spit venom at artificial intelligence on public forums and play the code purists are quietly, in the privacy of their offices, using AI to the max. They modify huge files with it, generate repetitive blocks of code, and clean up errors — because they're simply lazy and want to finish the job faster. Publicly, though, they'll defend their bastions to the bitter end. Why? For a simple reason: fear. They're afraid that a guy from the construction site, who has no diploma but has vision, iron logic, and can precisely direct artificial intelligence, can create in a few weeks a product their

six-person team would work on for half a year. Their fear bred envy.

My direct answer to the helplessness of the traditional approach to programming became SpamSlap. The contact forms on my sites were constantly bombarded by bots. SEO offers, databases flooded with junk; the lack of rate-limiting from my previous developers once drove my server to disaster and a blacklist. What did I do? Instead of installing huge, sluggish plugins, I wrote, with AI support, a lightweight snippet in Vanilla JS.

SpamSlap intercepts the form request before it even reaches the database. Using FastAPI and the lightning-fast Google Gemini Flash model via OpenRouter, the system analyzes the message content in a split second. If it's spam — the bot gets a multilingual, cynical refusal straight to the screen, and the server doesn't even flinch. Real emails pass through untouched. Simple? Brilliant. And created without the participation of a single "computer science professor."

Chapter 6: The Autonomous Empire

The real revolution isn't about mindlessly asking artificial intelligence to write yet another function. The revolution is about creating a multi-agent architecture that starts thinking and acting for you while you rest. When my developer token limits say "stop," I switch to marketing. But I don't do it by hand. That's what my own custom machines are for.

The first of them is **PostPilot** — a tool for automatically generating and scheduling social media content. Instead of paying corporations for expensive, flagship models, I did a solid cost optimization. I tied a cheaper, fast model for drafts to a stronger one for the final quality polish, and added a Human Error Mode — so the posts look like a living person wrote them, with small mental shortcuts and a natural tone. The system pulls specific data from my projects (Dynamic Data Extraction) and selects graphics through an API. The result? Running this machine costs me a fraction of what I'd pay if I pushed everything through flagship models.

My big, private machine is **ArticlePilot** — a system that runs my own blogs. It's a powerful content automation engine that takes care of my organic visibility online. The principles of its deep operation remain my sweet

secret. I can tell you, though, how this structure manages the process. I created a virtual Director of automation there, who constantly analyzes the web, checks historical articles so as not to duplicate topics, and studies people's reactions.

Because artificial intelligence has a tendency to hallucinate, above the Director I placed a second agent: a virtual Editor-in-Chief. When a text is generated, the Editor doesn't publish it blind. It writes out the claims it isn't sure of and sends them for verification — straight to the source of truth, meaning my own servers and the Brain system. Only when the facts come back confirmed does the text get corrected and cleared for publication. Nothing goes online until it's been checked. That's my answer to AI's greatest weakness: I don't let it guess where it can verify.

My Brain system greets me every day with a ready status of the work and questions about the things it isn't sure of. It searches its memory, fills in the missing data, and delegates the hard decisions to me.

All of this ties into my central server-fleet management platform — the Queen system. Queen works in hiding, on my server, communicating with the Sentinel instances. The Sentinel drones constantly monitor the health of my MySQL and PostgreSQL databases, track slow logs, and secure the hosts against cyberattacks. When they detect an anomaly, they send a report to

Queen, who analyzes the data in batches using a local LLM and sends the drones back an updated "knowledge book" with security guidelines.

```
[ Sentinel Drones ] ----(Reports on Bots/
Anomalies)----> [ Central QUEEN ]

^
|
        └─────────────(Updated Knowledge
Book)──────────────────┘
```

When I get up in the morning at 5:45, I no longer feel that old, destructive pressure that I have to program something by force at the cost of my health. I drink my coffee, I look at the monitor screen. Brain and my virtual agents did all the dirty work through the night. Everything is verified, tallied, ready to publish.

Artificial intelligence was supposed to replace people. In my world, I became its master. I stopped writing dry commands and brackets. I started thinking like an Architect, designing processes and making decisions. And that's the most important lesson of the modern web: it doesn't matter where you come from or whether you hold a diploma or a screwdriver in your hand. What matters is whether you can impose your own terms of the game on technology.