Documento: Prueba_traduccion_TLMBJpb_2

The Architect of Impossible Systems

In Silicon Valley there was a legend about a consultant who only appeared when a

project was destined to fail.His name was Daniel Ortega, and his specialty was

rescue software from the abyss.

The story begins on a Friday at 6 PM, when Laura Méndez, technical leader of

StartupHealth looked desperately at the architecture diagram of their system.They had

invested two years and five million dollars in a telemedicine platform that

It just didn't scale.Every time they exceeded 1,000 concurrent users, the

system collapsed.

"We need a miracle," he told the CEO.Or we rewrite everything from scratch, which

It will take another year, or we close.

That night, Laura received a strange email:

From: daniel.ortega@legacy-systems.io

Subject: I have seen your architecture.I can help.

Message: "Your problem is not technical. It is engineering. There is a difference."

The next day, Daniel arrived at the offices.He was a man of about 45 years old, with a gray beard,

with a laptop covered in stickers from conferences from the last 20 years.But the most

Impressive was the tranquility in his eyes, as if he had seen (and solved) all the

possible disasters.

"Show me your architecture," he said simply.

Laura displayed the diagram in the meeting room.It was a microservices monster: 47

different services, 23 databases, 8 message queues, 5 different frameworks, and a frontend that made direct calls to 15 different APIs.

Daniel watched in silence for five full minutes.Finally he spoke:

—This is not software architecture.This is the result of making technical decisions without understand the business problem.Let me guess: they started with a monolith, someone read about microservices on Medium, and decided to "modernize."

Laura blushed.It was exactly what had happened.

“Each service was created by a different team,” Daniel continued.There is not a coherent domain model.Your bounded contexts overlap.The service of "Patients" and "Users" share 80% of their logic.And let me guess: when There is a bug, no one knows which service is responsible.

—How do you know?

—Because I've seen this pattern hundreds of times.It's not your fault.It is the consequence of confuse tools with solutions.Microservices are a tool, not a goal.

Daniel opened his laptop and started drawing a new diagram.

—Software engineering is not about using the newest technology.It's about solving real problems with sustainable solutions.Check this out: its core domain is simple: Doctors, Patients, Appointments, and Records.Four entities.They don't need 47 services to model this.

Over the next few hours, Daniel taught them something they had forgotten in their haste to be "modern":

1. Domain-Driven Design is not optional

—Before writing code, they need to understand the domain.What is a "date" in your

context?Is it the same for the doctor as for the patient?How is it related to the

billing?These questions define your architecture, not the other way around.

2. Complexity is the enemy

—They have 23 databases because each team chose their favorite.PostgreSQL, MongoDB,

Redis, Cassandra, MySQL... —Daniel shook his head—.This is not architecture

polygota, it is technical anarchy.Each technology adds operational complexity: backups,

monitoring, expertise necessary.Do they really need all this?

3. Patterns exist for a reason

—They are using Event Sourcing for user login.Because?There is no need to

full audit there.And yet, they do not use it for medical records, where it is

It is critical to know who modified what and when.They are applying patterns because they sound cool, no.

because they solve problems.

4. Code is debt

—Every line of code they write is a promise of future maintenance.If something

It can be solved with configuration, don't write code.If something can be bought instead

If it is built, don't build it.Its core business is telemedicine, not building a

authentication system from scratch.

Laura and her team listened fascinated.It was like someone finally put in

words everything they had felt but did not know how to express.