

Rebuild the software you've run for years. Lose nothing on the way.

Plenty of businesses run every day on software written ten or fifteen years ago. Whoever built it is long gone, nobody knows everything it does, and every change feels risky. Rebuilding it is possible. **The real risk isn't the new code — it's everything the old system does that nobody ever wrote down.**

Developed rebuilding a decade-old multi-tenant business portal from scratch — procedural PHP and classic ASP, no framework, no documentation, no tests, and no one from the original build to ask. Client withheld.

SCREENS	DIVERGENCES	VERIFIED	REGRESSIONS
43 compared one by one, twice	146 found, written down, numbered	140 on the live build; 6 awaiting access	0 at final verification

THE PROBLEM NOBODY SEES

The obvious way to rebuild a system is to read its code and rewrite it. I did that first, and it wasn't enough: comparing the rebuild against the old system while it was actually running contradicted four conclusions drawn from the code and surfaced four more things the code couldn't show. The real rules of a business system live in how people use it every day.

So the new system isn't checked against what the old one should do. It's checked against what it actually does.

HOW I WORK

- 01 Old and new, side by side.** Same action, same data, both systems at once, screen by screen — as an administrator and again with your users' permissions, because some problems only show up from the lower role.
- 02 Every divergence is written down and numbered.** Nothing from memory: a precise register of what the old system does, what the new one does, and how it gets fixed.
- 03 Fix before moving on.** A few screens at a time, each corrected and re-checked before the next — with the test database restored between batches so nothing leaks from one check into another.
- 04 Re-check everything at the end.** Every item in the register, on the final build — not just the last ones fixed.

A DIVERGENCE IS NOT A NEW REQUEST

During a rebuild it's natural to ask for new things too. They go on a separate list. A divergence is something the old system did and the new one doesn't: that gets fixed, inside the agreed work. Something the old system never did is a new request, and it's priced separately. **That's how the quote stays the quote** — and how "done" keeps meaning done.

THE OLD SYSTEM STAYS ON

It's the question everyone asks: what if something goes wrong? The switchover is set up like this:

- The old system stays running and untouched until the new one is verified. The business doesn't stop.
 - Your data moves to the new system only after a check done together with you.
 - If you prefer, the screens stay the ones your staff already know — nothing to relearn.
 - Until you decide to switch, you can always go back.
-

SECURITY, FOUND WHILE WORKING

Walking every screen of an old system with an engineer's eye surfaces more than divergences. On that portal it surfaced nine security findings. None were exploited; all were reported in writing, confidentially and at class level only.

CLASS	SEVERITY	EVIDENCE
Credentials stored and displayed in cleartext	CRITICAL	Verified
Personal data reachable without authentication through an auxiliary path	CRITICAL	Verified
Authorisation trusted to client-modifiable cookies	CRITICAL	Deduced
Unauthenticated write endpoint; public diagnostic files exposing configuration	HIGH ×3	Verified
No login rate limit; uploaded documents protected only by URL secrecy	MEDIUM ×2	Mixed
Ancillary endpoints without authentication	LOW	Verified

For the client, that's a security review delivered as a by-product of the rebuild — before the switchover, not after an incident, and without a separate engagement to commission it.

And if something in the old system is dangerous, the new one doesn't reproduce it, however convenient. Passwords, for example: the client asked to keep them visible; in the rebuild they're hashed and nobody can read them.

WHAT YOU'RE LEFT WITH

Besides the new software, you get a document describing precisely what your system does, screen by screen and role by role. It's often the first one it has ever had. Whoever works on it next — me, another engineer, a test suite, or AI-assisted tooling — starts from there instead of from zero.

**Tell me what software you run and what worries you about it.
I'll tell you honestly whether to rebuild it, fix it, or leave it alone.**

Richard Trujillo • Desemax. Senior backend and AI-integration engineer. Eleven years of backend engineering, independent since 2015. Based in Milan, working across Europe.

richard@desemax.it · richardtrujillo.desemax.it