Practical guide

Legacy software risk for owner-managed firms

Why ageing systems that "still work" become a business risk for owner-managed UK firms — and how to reduce that risk with incremental modernisation instead of a reckless rewrite.

5 July 2026 · Legacy modernisation · Risk · Operations

Ageing software that "still works" can be the riskiest asset in the business — especially when only one person understands it.

If you run an owner-managed firm without a CTO, that sentence is usually familiar. The system ships orders, raises POs, or holds customer history. Nobody wants to touch it. Support gets dearer. Changes wait until something customer-facing fails. By then you are choosing under pressure, with only a vendor's voice in the room.

Legacy risk is not a developer problem first. It is an operational and leadership problem: fragile trading, concentrated knowledge, and fewer safe options every year you defer the conversation.

What "legacy" actually means in a real business

Forget the textbook. In practice, software is legacy when several of these are true:

- Nobody can explain the full process it encodes without the original builder
- The platform, language or hosting is unsupported or barely supported
- Small changes take weeks, cost a fortune, or are refused
- Integrations are held together with manual exports, email and spreadsheets
- You cannot get a clean export of the data you would need to leave
- Peak trading depends on "don't reboot anything" folklore

The system may still be profitable. That does not make it safe. It makes the risk quieter — until it is not.

Why owner-managed firms feel this harder

Without senior technology leadership in-house:

- Suppliers set the roadmap and the urgency
- Tribal knowledge substitutes for documentation
- "If it ain't broke" wins every budget round — until a key person leaves or a server dies

- A big-bang replacement looks like decisive leadership on a slide, and like a freeze on improvement in real life

You do not need to become technical. You do need a clear picture of what the system does for the business, what fails, and what must not break.

Typical risk patterns

Watch for these patterns; they often arrive together:

- **Unsupported platforms** — abandoned libraries, obsolete servers, licences nobody renews consciously
- **Key-person dependency** — one head holds passwords, quirks and the real process
- **Shadow processes** — the "official" system plus the spreadsheet that actually runs the day
- **Brittle integrations** — courier, accounts or webshop links that only one person dares restart
- **Deferred change** — every improvement parked until a failure forces a crisis project
- **Unknown data quality** — duplicates, silent gaps, and reports nobody fully trusts

If your sleep is already tied to one box in a cupboard or one contractor's phone, treat that as evidence, not personality.

Big-bang rewrites rarely help

A full replacement looks tidy: new vendor, new UI, "we'll migrate everything in three months".

In practice it often:

- Freezes improvement while the programme runs
- Underestimates edge cases the old system already handled for years
- Discovers data quality only after go-live pressure starts
- Trains people twice — once on workarounds, again on the new product
- Leaves you locked to a new supplier before the process is clear

Sometimes replacement is right. It is rarely right *first*, and almost never right as a panic response to the first outage.

A safer modernisation sequence

Think in risk reduction, not vanity rebuilds:

1. **Document what the system actually does** — in business language: orders in, stock moves, invoices out, exceptions
2. **Name what must not break** — the thin slice of capability that keeps trading alive
3. **Separate containable risk from existential risk** — backups and access first; architecture second
4. **Stabilise** — hosting, backups, credentials, monitoring, who to call at 6am
5. **Carve a thin improvement** — one integration, one module, one workflow — proven on real data
6. **Migrate incrementally** — with rollback, knowledge transfer and a written exit for your data
7. **Only then decide** — improve, wrap, replace, or retire — against outcomes, not demos

This is slower on a slide. It is usually faster at keeping the business trading.

What to write down before you call anyone

A useful one-page brief beats a vague "we need to modernise":

- What the system is for (three sentences)
- Who uses it daily, and who is the only person who truly understands it
- Where it fails or scares you (be specific)
- Which reports or integrations are trusted
- What a bad week would look like if it died on Monday
- What "good enough in twelve months" means in operational terms

If you cannot write that page, you are not ready to buy a rewrite. You are ready to discover.

Questions worth asking suppliers and advisers

- Can we reduce the worst risk without replacing everything this year?
- How do we get a full export of our data, and in what form?
- What knowledge will transfer to our team — or are we recreating a single point of failure?
- What is the rollback if a migration step fails mid-trading?
- Which edge cases in the old system are we at risk of losing?
- Are you recommending a product you resell, or options scored against our outcomes?

Prefer people who will say "not yet", "stabilise first", or "keep this part". That honesty is rarer than a shiny demo.

Improve, wrap, replace, or retire

Not every legacy system deserves the same fate:

- **Improve** — when the core fits and risk is mostly hosting, access, or a few brittle links
- **Wrap** — when you keep the engine but present or integrate more safely around it
- **Replace** — when the product genuinely cannot support how you trade now, and a proof on real data agrees
- **Retire** — when a spreadsheet, a simpler tool, or a quieter process already does the job better

The wrong call is treating every ageing system as a full replacement project. The other wrong call is waiting until the only option left is emergency replacement.

A note from long-lived systems

The best operational software earns its keep by encoding real judgement — purchasing rules, pick logic, exception handling — not by looking modern. See the automated purchase ordering case study: years in continuous use because it replaced a manual process that depended on one person's knowledge, and kept doing that job.

Longevity is a feature when knowledge is shared and risk is managed. Longevity is a trap when nobody dares touch the thing and nobody else could rebuild it.

Where Two Wrens helps — if you want a partner

Legacy modernisation at Two Wrens is discovery, architecture review and careful incremental migration — with knowledge transfer so the business is not stuck again. Deeper or messier situations often start as Technology Consultancy.

Those are options. The point of this guide is clearer risk and a safer sequence — whether you work with us, another adviser, or your existing team.

Next step

Book a Discovery Call if legacy risk is already affecting trading, insurance conversations, or sleep. Bring the one-page brief above if you can. If you cannot, that is a useful starting point on its own.

Talk it through

If this guide is useful, a Discovery Call is the natural next step — no obligation, no sales script.

Book: twowrens.co.uk/contact
Email: hello@twowrens.co.uk