QOS SOFTWARE Development

## MODERNIZE OR REPLACE?

# AN HONEST LEGACY SOFTWARE CHECKLIST

877.800.7672

qossoftware.com

# Modernize or Replace? An Honest Legacy Software Assessment Checklist

Every aging business system reaches a point where someone asks the hard question: do we keep patching this, rebuild it in pieces, or throw it out and start over? It's an expensive decision to get wrong in either direction — and it's usually made on gut feel, vendor pressure, or the frustration of the last outage rather than a clear look at the system itself.

This checklist is that clear look. Work through it before you commit to modernizing or replacing anything. It has no prices in it on purpose: the decision comes first, and cost follows the decision. The goal is to walk into any conversation — with your team or with a development shop, ours included — knowing what your system is really worth keeping, and what it will cost you to keep pretending it's fine.

## 1. Start With What the System Actually Does

Before you judge a legacy system, write down what it truly does for the business today — not what it was built for years ago, but the work that depends on it right now.

### Answer First:

- What core operations run through this system every day?
- What would stop working tomorrow if it went dark?
- Which parts are genuinely used, and which are dead features nobody touches?
- Is this system a competitive advantage, or plumbing you could replace with a product?

### The Rule:

You can't decide the future of a system you can't describe in the present. Get its real job down to a short list before you weigh anything else.

## 2. Assess What Still Works — and What Doesn't

Legacy is not the same as broken. Plenty of old systems do their core job perfectly well; the problems are usually at the edges. Separate the two honestly.

### Sort It Into Three Piles:

- What works reliably and rarely causes trouble?
- What is fragile — breaks under load, needs manual workarounds, or fails at month-end?
- What is simply impossible now — changes you can't make, reports you can't get?
- How often does someone say "the system won't let us do that"?

### Why It Matters:

The reliable core is worth preserving and often worth modernizing around. The impossible pile is what actually forces the decision. Knowing which is which keeps you from replacing something that mostly works.

## 3. Judge Your Data Quality and Portability

In most legacy systems, the data is worth more than the code. Whether you modernize or replace, that data has to come with you — so its condition drives the whole project.

### Look Hard At:

- How clean is the data — duplicates, gaps, fields used for the wrong purpose?
- Can you get it out in a usable form, or is it locked in a proprietary format?
- Do you understand the structure, or has the meaning of the tables been lost over time?
- How much history must move, and how much can be archived instead?
- Who would validate that migrated data is correct?

### The Rule:

Migration risk lives in the data, not the features. Messy, undocumented data is the single most common reason a "simple" replacement turns into a long one.

## 4. Map the Integration Debt

An old system rarely stands alone — it's wired into other tools by connections that may be undocumented, manual, or held together with a spreadsheet. Surface every one before you decide anything.

### Inventory the Connections:

- What other systems send data to or pull data from this one?
- Which connections are automated, and which are someone re-keying data by hand?
- Are those links documented, or do they live in one person's memory?
- What breaks downstream if this system changes shape?
- Which of these integrations would have to be rebuilt either way?

### Why It Matters:

Integration debt is the hidden cost that ambushes replacement projects. A clear map turns a set of surprises into a set of tasks you can plan and price.

## 5. Weigh the Staffing and Knowledge Risk

A system is only as maintainable as the people who understand it. Legacy risk is often less about the technology and more about who is left who can keep it alive.

### Be Honest About:

- Who understands how this system works — and how close are they to leaving or retiring?
- Is the technology it's built on still supported, or has it reached end of life?
- Can you still hire people who know this platform, or is the talent pool gone?
- Is the original vendor still around and still shipping updates?
- What happens the day the one person who knows it walks out?

### The Bottom Line:

A system nobody can maintain is a system already on borrowed time. When the knowledge risk is high, doing nothing is not a neutral choice — it's a bet the business may not want to make.

## 6. Decide: Modernize in Phases, or Replace

With the first five sections answered, the real decision comes into focus. It's rarely all-or-nothing — the best answer is often to modernize the parts worth keeping and replace the parts that aren't.

### Lean Toward Modernizing When:

- The core logic still fits how the business actually works.
- The data is understood and reasonably clean.
- You can improve it in stages without a risky overnight cutover.

### Lean Toward Replacing When:

- The system fights the business more than it serves it.
- The platform is unsupported and the knowledge is gone.
- A packaged product now covers what used to be custom — and the process isn't your edge.

### The Rule:

Phased modernization is usually lower-risk than a full replacement, because it keeps the business running while the system changes underneath it. Reserve the full rebuild for when the old system is genuinely beyond saving.

## 7. Plan the Transition Without a Shutdown

Whichever way you go, the operation has to keep running during the change. The transition plan is where good projects separate from painful ones.

### Plan For:

- Can the old and new systems run side by side while you cut over piece by piece?
- What is the rollback plan if a phase goes wrong?
- How will you keep data in sync during the overlap?
- Who trains the team, and when — before, not after, go-live?
- What is the smallest first phase that proves the approach works?

### Why It Matters:

A big-bang shutdown is where legacy projects go to fail. A phased plan with a rollback at every step turns a bet-the-business event into a series of manageable steps.

## 8. Define What “Better” Looks Like

If you can’t say what a successful outcome is, you can’t tell whether the project was worth it. Define “better” in plain, checkable terms before anyone starts building.

### Set Your Measures:

- What will be faster, cheaper, or finally possible that isn’t today?
- Which of today’s daily frustrations must disappear?
- What does “good enough to switch” look like versus “everything we ever wanted”?
- Which capabilities are must-have, and which are nice-to-have?
- When will you review whether the change actually delivered?

### The Bottom Line:

A must-have list and a nice-to-have list are the two most valuable documents in any modernization. Write them before you ask anyone for a proposal.

# The Final Word

The modernize-or-replace decision is only as good as the assessment behind it. Answer these questions honestly and you’ll know whether your legacy system is an asset worth rebuilding around or a liability worth retiring — and you’ll be able to tell whether a development shop is solving your problem or selling you a project.

## Next Step:

Start with sections 1, 3, and 5 — what the system does, the state of your data, and who still understands it. Those three decide most of the outcome. When you’re ready to turn the assessment into a plan, QOS Software will scope it with you honestly, tell you what’s worth modernizing versus replacing, and say where a packaged product would serve you better. Call **877.800.7672** or visit **qossoftware.com**.