broadpin



Software Modernization and Its Cognitive Biases

Why legacy modernization projects fail differently from new developments, and what to do about it

Why Modernization Follows Different Rules

Modernization projects rarely fail because of technology alone. They fail because teams and leaders underestimate how strongly perception, experience, and old patterns of success shape decisions before the first line of new code is written.

The root cause is a category error: treating modernization like greenfield development. New software starts from an open solution space. Modernization starts inside an existing fabric of code, data, habits, tacit knowledge, and earlier decisions. That difference plays out in every phase of the project. Scope is defined implicitly ("everything as before, but better"). Original business needs are buried under years of staff turnover and missing documentation. Users skip the vision and jump straight to screens and buttons. Interfaces and integrations have to be considered from day one. And the IT department holds critical knowledge that no workshop will surface on its own.

These differences create decision-making situations in which cognitive biases take effect with particular ease. The familiar functional scope is underestimated, old requirements are taken for granted, missing documentation is replaced by assumptions, and technical dependencies only surface after effort and budget have already been committed.

This whitepaper identifies the biases that most reliably derail modernization initiatives and shows how a structured, analysis-first approach neutralizes them in practice.

The Biases That Derail Modernization

Starting wrong

The Turkey Illusion. Teams follow the natural-seeming principle of "from simple to complex," gain confidence sprint by sprint, and assume the hard parts will follow the same curve. They do not. Around month four, a cluster of tightly coupled legacy logic appears that nobody analyzed upfront, and the architecture built on simple cases cannot absorb it. In a recent Broadpin engagement, the decisive question was not "What is easiest?" but "What gives us the most momentum?" That required knowing what the system actually contained before making the first move.

Confirmation Bias. When reverse-engineering the legacy application, it is nearly impossible to start with a neutral lens. Every time the thought "I knew it!" occurs while reading existing code, the team has stopped analyzing and started confirming. Edge cases, implicit business rules, and undocumented dependencies slip through, only to resurface months later.

The Bandwagon Effect. The first question in most modernization projects is "What will we build on?" and the decision is often made before the question has properly been asked. A technology is trending, a team lead champions it, and the project is shaped by preference rather than by evidence from the legacy system's actual requirements.

Estimating wrong

Scale Insensitivity. In one modernization project, a developer responsible for UI data binding said: "I only need ten minutes to rewrite this event. I would rather do it manually." A fair point, in isolation. Unfortunately, the legacy application contained more than 4,000 places that needed the same change. At ten minutes each, that person would have spent over half a year doing nothing but copying code. In agile delivery, work arrives sprint by sprint. Modernization delivers differently: if the legacy application has not been inventoried, millions of lines of code land on the board at once. Every individual decision multiplies with the full complexity of what already exists.

The Planning Fallacy. Because the existing system already works, it creates the impression that the new solution merely has to be reconstructed. The original wrong decisions, rework cycles, and debugging sessions that went into making it work are invisible. Running systems look easy to replicate. They are not.

Overconfidence. Especially with in-house applications, the business department's expectations grow while the budget does not allow for external expertise. IT responds with "we will manage," even though risks, dependencies, and missing specialist knowledge have not been sufficiently understood. When overconfidence meets scale insensitivity meets the planning fallacy, the result is a project plan with no realistic foundation underneath it.

Remembering wrong

The Peak-End Rule. Teams that built the legacy application remember two moments: the peak (releases two and three, when everything worked) and the end (the current state, which largely meets requirements). Forgotten are the failed attempts, the redesigns, and the long debugging phases. That selective memory is why optimistic assessments gain the upper hand: "Of course we will manage it. The software is running, so why should we have problems?"

Survivorship Bias. A significant proportion of modernization initiatives fail to meet their time, budget, or scope targets. That reality does not match the polished success stories on social media and in vendor keynotes. Teams plan against examples that survived, not against the full distribution of outcomes. The project plan ends up reflecting what worked for someone else, not what the actual legacy system demands.

Hindsight Bias. The legacy application looks like the result of a clear plan. It was not. It was the result of iteration, compromise, and accumulated fixes. Modernization inherits all of that complexity, visible or not.



From Bias to Method

None of these biases are character flaws. They are normal shortcuts the human brain takes under complexity and uncertainty. The question is whether the project structure accounts for them. At Broadpin, we have learned that the technical migration is only half the challenge. The other half is making sure decisions are grounded in what the legacy system actually contains, not in what the team assumes it contains.

Our approach starts with analysis, not with architecture. PITSS.CON's Application Data Cube provides the factual baseline that these biases systematically prevent teams from building on their own:

- **Against Starting Wrong.** The dependency graph shows real complexity upfront. The team starts with the cluster that delivers momentum, not the one that feels safe. Technology choices follow evidence, not trends.
- **Against Estimating Wrong.** The automated inventory quantifies the actual size of the task: not "roughly 200 screens" but "4,127 UI data bindings, 312,000 lines of PL/SQL, 1,847 table APIs." Estimation becomes an engineering task, not a guessing exercise.
- **Against Remembering Wrong**. Reverse engineering surfaces the forgotten complexity: edge cases, implicit business logic, undocumented dependencies. The analysis shows what the system actually contains, not what the team remembers building.

Only after the assessment is complete does the team move to architecture and planning. By that point, the scope is defined by evidence, the effort is estimated against measured complexity, and the starting point is chosen for strategic reasons rather than for comfort.

© Broadpin I broadpin.com

Next Steps

Modernization is less a purely technical issue and more a manageable leadership task. The window to address cognitive biases is at the beginning, when the analysis shapes the plan rather than the plan shaping the analysis. That is where the most consequential decisions are made, and that is where getting them right matters most.

If you are planning a modernization, or already in one and things feel harder than expected, talk to the Broadpin modernization team. We bring structured analysis, proven methodology, and direct experience with large-scale legacy modernization to every engagement.



Ready to modernize on facts instead of assumptions?

Book a free modernization assessment. We'll analyze your legacy application with PITSS.CON, quantify its actual size and complexity, map the dependencies that will shape your migration, and outline a realistic scope, starting point, and timeline before a single architectural decision is made.

→ Schedule your free consultation today.



broadpin

Your solution is here.