CLOUD · DIGITAL MODERNIZATION

Modernize Like You Mean to Keep It

Why big-bang rewrites fail mission systems — and how incremental modernization delivers value early while retiring the legacy that is holding you back.

 Intelligence. Integrity. Impact.

Cloud · Digital Modernization

Most modernization programs are sold as a destination: a shiny new system, live on a distant date, replacing everything at once. The graveyard of federal IT is full of those programs. They ran for years, spent the budget, delivered late or not at all, and left the legacy system running anyway. There is a better pattern, and it starts by refusing the big-bang premise.

The legacy tax

~80%

Share of federal IT spending that goes to operating and maintaining existing systems, not improving them.

GAO

10

Critical federal legacy systems GAO singled out as most in need of modernization — several decades old.

GAO, 2019

Big bang

The all-at-once rewrite pattern that most often fails — and the one this paper argues against.

Artheus view

The trap you are already paying for

Legacy systems are expensive precisely because they work well enough to keep running. Roughly four of every five federal IT dollars go to operations and maintenance rather than modernization, according to the Government Accountability Office, and some of the government's most critical systems are decades old, running on languages and hardware their agencies can barely staff. Every year of deferral raises the cost and the risk of the eventual change. The question is never whether to modernize — it is how to do it without betting the mission on a single distant date.

Why the big-bang rewrite fails

A full replacement concentrates every kind of risk into one event. Requirements drift over the years it takes to build. Value arrives last, if at all, so there is nothing to show and nothing to correct against until the end. And the cutover is all-or-nothing on a system that cannot afford downtime. GAO's own catalog of troubled IT programs is, in large part, a catalog of big-bang attempts. The pattern does not fail because the teams are weak; it fails because the shape of the bet is wrong.

Value delivered: two ways to modernize

The difference between the two approaches is not the destination — it is when value shows up and how much is at risk at any moment. Incremental modernization delivers working improvements early and often; the big-bang approach delivers nothing until a high-risk finale (Figure 1).

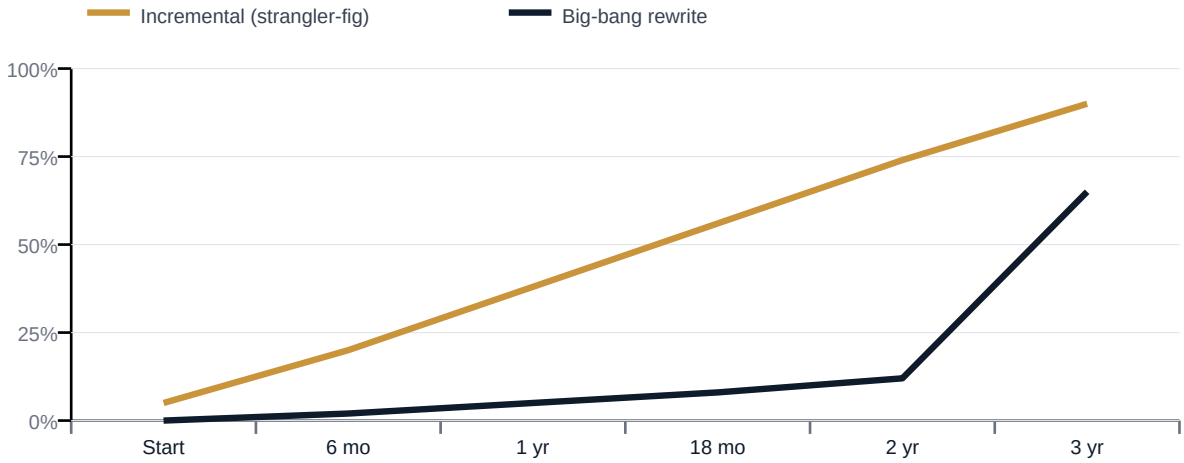

Figure 1. Illustrative pattern. Incremental modernization banks value continuously and keeps risk small at every step; the big-bang rewrite delivers nothing until a single high-risk cutover — the point at which many programs stall or fail.

The strangler-fig pattern

Named for the vine that grows around a tree and gradually replaces it, the strangler-fig approach modernizes one capability at a time. You wrap the legacy system behind a modern interface, route a single slice of traffic to a new implementation, prove it in production, retire that slice of the old system, and repeat. The legacy shrinks as the new system grows, and at no point is the mission one failed cutover away from disaster (Figure 2).

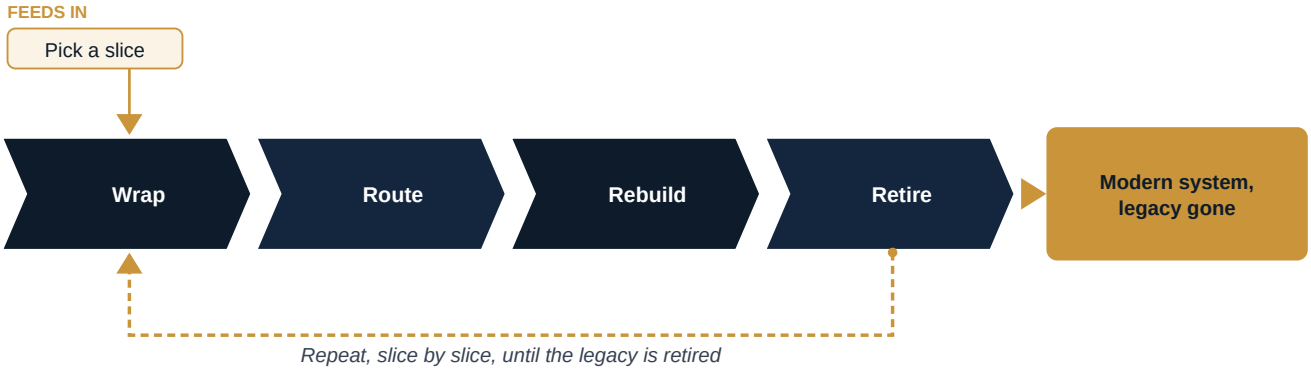

Figure 2. Modernization as a repeatable loop, not a single event. Each pass takes one capability from the legacy system, rebuilds and proves it in production, and retires the old piece — so value is continuous and risk stays bounded.

What to modernize first

Not every slice is worth taking first. The best opening move is the one that is painful, bounded, and visible — a capability whose users feel the legacy pain daily, whose scope is small enough to finish in months, and whose improvement is obvious enough to build support for the next slice. Avoid starting with the hardest, most entangled core; that is where big-bang programs go to die. Start where a quick, real win proves the pattern and earns the room to keep going. Momentum, not heroics, is what carries a modernization through to the end.

Cloud-smart, not cloud-first

Moving to the cloud is a means, not the goal. Lifting a legacy system unchanged into a cloud account — the classic “lift and shift” — often just relocates the technical debt and the bill. The discipline is to match each workload to where it belongs: some to a FedRAMP-authorized cloud service, some re-architected, some left on-premises for sovereignty or latency reasons (see our companion paper on in-boundary AI). Modernization is a portfolio decision, made workload by workload, not a mandate to move everything.

Govern the change, and keep the off-ramps

Incremental modernization needs its own guardrails: clear ownership of each slice, acceptance criteria before a slice is cut over, and honest measurement of what the new implementation actually improved. Just as important, avoid trading one lock-in for another. Favor open interfaces and portable data so that the modern system you are building does not become the legacy trap of the next decade. Modernize like you mean to keep it — because you will.

Start small, deliver early, retire as you go

The organizations that modernize successfully are not the ones with the biggest programs. They are the ones that picked a painful slice, delivered a real improvement in months, proved the pattern, and repeated it until the legacy was gone. That path is less dramatic than a grand replacement and far more likely to end with a system people actually use — and a legacy system finally switched off.

References

1. U.S. Government Accountability Office, *Information Technology: Agencies Need to Develop Modernization Plans for Critical Legacy Systems*, GAO-19-471, 2019.
2. U.S. Government Accountability Office, reporting on federal IT operations-and-maintenance spending as a share of the IT budget.
3. Office of Management and Budget, *Federal Cloud Computing Strategy (Cloud Smart)*.
4. FedRAMP — Federal Risk and Authorization Management Program, program documentation.
5. M. Fowler, *StranglerFigApplication* (the incremental-replacement pattern).
6. National Institute of Standards and Technology, *Zero Trust Architecture*, NIST Special Publication 800-207, August 2020.