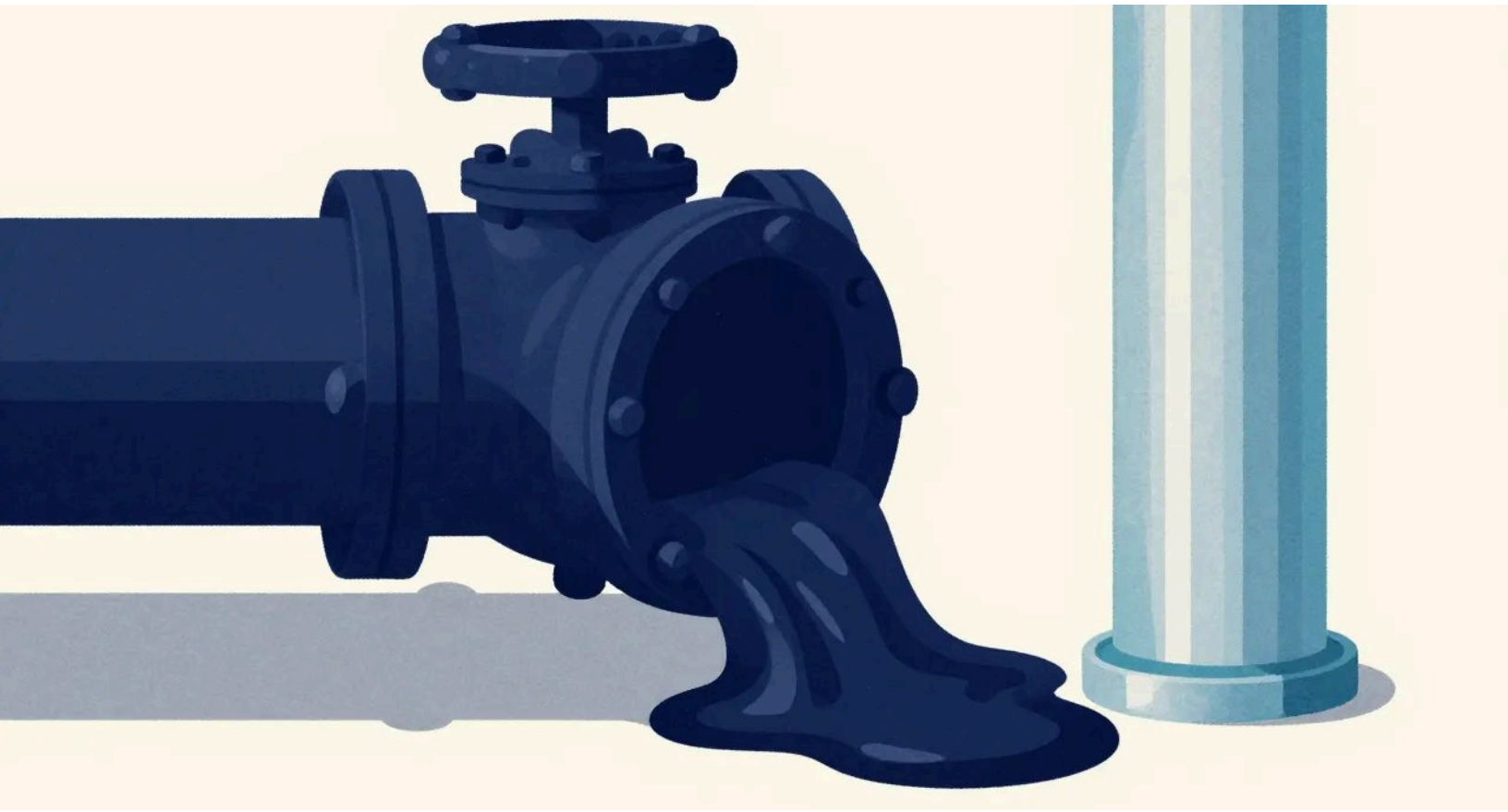


EXECUTIVE PAPER · SYSTEMS RESCUE

Stop the Bleeding Before You Modernize

When production is failing, a rewrite looks like the cure. The cheaper, safer sequence is to stabilize, fix root causes by risk, and then modernize.

AUTHOR

Chris Watkins

SERVICE

Systems Rescue

FOR

CEO, CTO · 12 min read

AT A GLANCE

The problem. A system that fails every week invites a bold answer: replace it. Leadership approves a rewrite or a new platform while the old system keeps breaking, and the team now carries two loads. Recent public cases show replacements that went live unstable and cost several times their budget.

Why it persists. A rewrite is easier to fund than repair work, because it promises a clean finish and a press release. Teams in constant firefighting rarely know which failures cost the most, so they cannot argue for a specific fix. Most human-error outages trace to process and procedure, which new technology does not change.

The recommendation. Treat stabilization as a gate. Stop the bleeding, rank failures by business risk, and fix root causes in that order with every change reversible. Fund modernization only after the current system meets written stability criteria for a sustained period. The work of stabilizing produces the knowledge a successful modernization needs.

The Decision in Front of You

The board has heard about the outages, and customers have complained. Someone has proposed a fix: a new platform, a cloud migration, or a rewrite that retires the problem system in 18 months.

The decision is about order. You can fund the replacement now and keep the old system limping along, or you can spend the next 90 days making the current system stable and then decide what to modernize, with evidence.

The stakes are measurable. Uptime Institute's 2026 outage analysis found 57% of respondents said their most recent major outage cost more than $100,000, and one in five put it above $1 million [1]. A replacement project does nothing to reduce that cost until cutover.

What's Happening

Most outages come from how work is done

Uptime's 2025 analysis found nearly 40% of organizations had suffered a major outage caused by human error in the previous three years. Of those incidents, 85% came from staff not following procedures or from flaws in the procedures themselves [2]. In 2026, failure to follow established procedures was again the leading driver of human-error outages [1]. Coverage of the 2025 report noted that about 80% of operators believed better management and processes might have prevented their most recent major outage [3].

A new platform inherits the organization's change process, its runbooks, its alerting, and its habits. If those caused the outages, the outages migrate with the workload.

Instability blocks the teams that would do the modernizing

Google's DORA research program surveyed nearly 5,000 technology professionals for its 2025 report. Its central finding is that new tools amplify whatever a team already has, so strong teams gain and struggling teams see their problems grow [12]. Tightly coupled systems with slow feedback loops saw little or no benefit from AI tools, and AI adoption remained associated with lower delivery stability [12].

The same report sorted teams into seven profiles. One, the "legacy bottleneck," describes teams that are always reacting because unstable systems set their agenda; it accounted for 11% of respondents. Another 10% were in "foundational challenges," described as survival mode [13]. DORA's 2024 report found that unstable priorities reduce productivity and raise burnout, and that internal platforms can lower change stability during adoption [14]. A platform program handed to a team in survival mode adds both.

Recent cases: modernization that became the incident

Sonos, 2024. Sonos had embarked on what its CEO called a complete ground-up rewrite of its mobile app, intended to fix performance and reliability problems that had built up over roughly 20 years [7]. The new app shipped in May 2024 with serious bugs and missing features, and the company could not roll it back [8]. Sonos told investors it expected to spend $20 million to $30 million in the short term to win back customers and partners, and it delayed two major product launches that were ready to ship [7]. It later said it would miss its annual revenue target by $200 million because of those delays [8]. The CEO stepped down in January 2025 [9]. Among the company's public commitments afterward was an end to all-at-once app releases [8].

The rewrite targeted real reliability debt. Because it shipped as one large change with no way back, the cure for instability became a customer-facing failure of its own, and the company spent the following year repairing it while new revenue waited.

Birmingham City Council, 2022 to 2026. The council replaced its SAP finance system with Oracle, initially estimated at £19 million plus £1 million contingency, and went live in April 2022 [10][11]. By January 2026 the forecast program cost through 2027/28 was £144.4 million, the system was still incomplete, and the council was reimplementing Oracle from scratch [10]. Fraud-detection audit trails were off for more than 18 months, £2 billion in transactions landed in the wrong financial year, and manual workarounds cost more than £5 million [10]. The failure contributed to the council's effective bankruptcy in 2023 [10].

The auditors' account is the useful part. Testers had reported instability and their concerns were not given priority. Cash management was reported Green while bank file loading was failing. The auditors concluded that a clear presentation of test quality would likely have stopped the go-live, and that the

council could have delayed and kept supporting SAP [11]. The council had no written definition of "stable enough to cut over," so optimistic status reports filled the gap.

Why the Usual Approach Fails

Rewrites get funded because a replacement has a name, a vendor, and an end date, while repair work looks open-ended.

The team proposing the rewrite usually cannot say which failures cost the most. Firefighting consumes the hours that would go to measurement, so the organization has a strong feeling that the system is bad and weak data on why. A rewrite approved on that basis reproduces the failures nobody diagnosed.

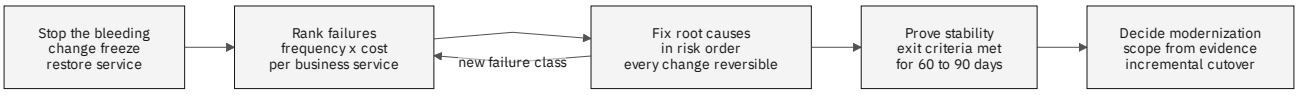
Running two systems at once doubles the load on the engineers who know the old one best, since they carry the pager and also explain its edge cases to the new project. The old system usually gets less care, fails more, and strengthens the case for a faster cutover.

Cutover pressure then overrides evidence. Birmingham's steering committee approved go-live on reports that understated failing tests [11]. Sonos shipped an app its own staff had concerns about [8]. In both cases the old system's instability, and the cost of the project already spent, made waiting feel more dangerous than going live.

The Framework: The Stabilization Gate

Google's SRE guidance for an outage starts with triage: make the system work as well as it can under the circumstances before hunting for root cause, because users gain nothing if the system dies during the investigation [4]. The same logic applies across a portfolio of failing systems: restore service, find out what is actually costing money, and decide on replacement once the evidence is in.

Exhibit 1. The Stabilization Gate: modernization is funded only after the current system proves stable



Source: author's framework, adapted from Google SRE triage and error budget practice [4][6].

Stop the bleeding. Freeze nonessential changes on the failing system. Google's example error budget policy halts releases other than urgent and security fixes until a service is back within its reliability target [6]. Restore service with the simplest safe measure, such as a rollback or added capacity, and label it temporary.

Rank failures by risk. List every failure mode seen in the past six months and any known exposure that has not yet failed, such as an untested backup. For each, estimate frequency per year and cost per occurrence. Multiply. Sort. This list replaces opinion with an ordered work queue.

Fix root causes in risk order. Each fix gets a written postmortem with an owner and a preventive action, following the triggers Google's SRE book uses for when a postmortem is required [5]. Every change ships with a tested way to undo it.

Prove stability. Agree on exit criteria before the work begins: for example, no severity-one incident in 60 days, change failure rate (the share of changes that cause a production problem) below an agreed threshold, and a restore of critical data tested in the last quarter. The criteria are a gate. Modernization funding waits for them.

Decide modernization from evidence. By now the team knows which components fail, which are merely old, and which carry hidden dependencies. Modernization scope shrinks to the parts that need it, and cutover happens in stages that can be reversed.

What risk ranking looks like

The following example uses a hypothetical regional distributor with one aging order platform. Substitute your own figures.

Exhibit 2. The highest-risk item is often the one that has not failed yet

Failure mode	Events per year	Cost per event	Annual exposure	Fix effort
ERP backup never test-restored; total data loss	0.1	$2,000,000	$200,000	2 weeks
Database volume fills, nightly batch fails	10	$15,000	$150,000	1 week
Partner EDI certificate expires unnoticed	2	$60,000	$120,000	2 days
Order API memory leak forces restarts	24	$4,000	$96,000	3 weeks
Total			**$566,000**	**about 8 weeks**

Source: author's illustrative estimate. Events per year and cost per event are assumptions for a hypothetical distributor; annual exposure is events per year multiplied by cost per event. Fix effort assumes one senior engineer with access to the system.

About eight weeks of focused work addresses $566,000 a year of exposure, long before any replacement could go live. The top-ranked item has never caused an outage, which is why teams in firefighting mode never reach it.

Costs, Risks, and Tradeoffs

Stabilizing first is a real cost. It delays visible progress and can frustrate a board that wants a transformation story.

Exhibit 3. Tradeoffs of stabilizing before modernizing

Area	Cost or risk	Mitigation
Time to modernization	Replacement starts 60 to 120 days later than planned	Use the stabilization period to document the system and write evidence-based requirements
Spend on a system you may retire	Money goes into a platform with a limited future	Fix only by risk rank; stop when exit criteria are met
Change freeze	Business requests wait during the freeze	Keep the freeze short and publish its exit criteria
Vendor and contract timing	A signed replacement contract may carry penalties for delay	Negotiate start dates that follow the stability gate
When this is wrong	The platform is past vendor support with no security patches, the hardware cannot be replaced, or a regulator has set a deadline	Run a short stabilization sprint on the top-ranked risks only, then proceed with a staged cutover and a tested rollback

Source: author's analysis.

Some systems cannot be stabilized at reasonable cost, because the knowledge to maintain them is gone. The risk ranking will show this when most of the exposure sits in components nobody can safely change. Replacement is then justified, and the stabilization work still yields the requirements and test cases it needs.

How to Act

Exhibit 4. A one-year roadmap from firefighting to funded modernization

Window	Actions	Owner	Milestone
First 30 days	Freeze nonessential changes on failing systems. Restore service with temporary measures. Build the risk-ranked failure list with frequency and cost. Write exit criteria for the stability gate.	CTO with operations lead	Ranked list of failure modes with annual exposure; exit criteria approved by CEO and CTO
First 90 days	Fix the top-ranked root causes, each with a postmortem, an owner, and a reversible change. Test-restore critical data. Track severity-one incidents and change failure rate weekly.	Engineering leads	Top five risks closed or mitigated; severity-one incidents down at least 50% from the 30-day baseline; restore tested
Day 90 to 180	Hold the exit criteria for 60 to 90 days. Document the system's dependencies and failure history. Draft modernization scope from the evidence.	CTO with architecture lead	Exit criteria met for a full period; modernization scope and business case based on measured data
First 365 days	Begin modernization in stages, each with a parallel run and a tested rollback. Keep the stabilized system's postmortem process running throughout.	CTO with program sponsor	First stage live with no severity-one incident attributable to cutover; old system retired only after the new one meets the same exit criteria

Source: author's recommendation.

Questions to Ask Your Team

1. Which failures cost us the most in the last six months, in dollars, and how do we know?
2. What is the worst failure that has not happened yet, and when did we last test our recovery from it?
3. How many hours a week do the engineers who know the current system spend firefighting?
4. If we approve the replacement today, who carries the pager for the old system until cutover?
5. What written criteria would tell us the current system is stable enough to begin a migration?
6. Can every change we ship this quarter be undone, and how long would undoing it take?

Closing

A failing system is a cost the business pays every month, and a rewrite approved in that state starts with an unknown list of requirements and a team that has no time. Stabilize the current system, rank its failures by what they cost, fix them in that order, and prove stability against written criteria. Then fund

modernization for the parts that need it, using what the repair work taught you.

Sources

[1] Uptime Institute. "Uptime Announces Annual Outage Analysis Report 2026." Press release via Business Wire. 2026. https://www.businesswire.com/news/home/20260513289344/en/Uptime-Announces-Annual-Outage-Analysis-Report-2026.

[2] Uptime Institute. "Uptime Announces Annual Outage Analysis Report 2025." Press release via Business Wire. 2025. https://www.businesswire.com/news/home/20250506733553/en/Uptime-Announces-Annual-Outage-Analysis-Report-2025.

[3] The Register. Coverage of Uptime Institute Annual Outage Analysis 2025, human error findings. 2025. https://www.theregister.com/2025/05/07/human_error_a_factor_in.

[4] Beyer, B., Jones, C., Petoff, J., and Murphy, N. R., eds. *Site Reliability Engineering: How Google Runs Production Systems*, Chapter 12, "Effective Troubleshooting." O'Reilly. 2016. https://sre.google/sre-book/effective-troubleshooting/.

[5] Beyer, B., Jones, C., Petoff, J., and Murphy, N. R., eds. *Site Reliability Engineering: How Google Runs Production Systems*, Chapter 15, "Postmortem Culture: Learning from Failure." O'Reilly. 2016. https://sre.google/sre-book/postmortem-culture/.

[6] Beyer, B., Murphy, N. R., Rensin, D., Kawahara, K., and Thorne, S., eds. *The Site Reliability Workbook*, Appendix B, "Example Error Budget Policy." O'Reilly. 2018. https://sre.google/workbook/error-budget-policy/.

[7] Sonos, Inc. Q3 fiscal 2024 earnings call transcript, via The Motley Fool. 2024. https://www.fool.com/earnings/call-transcripts/2024/08/08/sonos-sono-q3-2024-earnings-call-transcript/.

[8] MacRumors. "Sonos Promises to Recommit to Software Quality and Customer Experience After App Disaster." 2024. https://www.macrumors.com/2024/10/01/sonos-customer-service-software-quality/.

[9] CE Pro. "Sonos CEO Patrick Spence Steps Down as Company Looks to Get Back on Track." 2025. https://www.cepro.com/news/sonos-ceo-patrick-spence-steps-down-as-company-looks-to-get-back-on-track/145588.

[10] The Register. "Birmingham City Council's Oracle ERP fiasco now £144M and still not working." 2026. https://www.theregister.com/2026/01/29/birmingham_oracle_latest/.

[11] The Register. "How mega city council's failure to act on Oracle rollout crashed its financial controls." Coverage of Grant Thornton auditor's report. 2025. https://www.theregister.com/2025/02/27/birmingham_oracle_auditors/.

[12] Harvey, N., and DeBellis, D. "Announcing the 2025 DORA Report: State of AI-Assisted Software Development." Google Cloud Blog. 2025. https://cloud.google.com/blog/products/ai-machine-learning/announcing-the-2025-dora-report.

[13] Cusy. "DORA Report 2025: The state of AI-supported software development." Coverage of the DORA 2025 team profiles. 2025. https://cusy.io/en/blog/dora-report-2025.html.

[14] DORA (Google Cloud). "2024 Accelerate State of DevOps Report." 2024. https://dora.dev/report/2024.

The views in this paper are the author's own and do not represent any employer. Scenarios are hypothetical. This paper is not legal, financial, or compliance advice.