

Exposure Control and Experimentation

Chris Timperley · Monday, October 19, 2026

Administrivia

- **Midterm results** should be posted later tonight
- **Homework:** we will begin traffic simulator early this week
  - you can still build out your dashboard for Checkpoint A
  - we will likely reach out to ask you to configure certain settings (e.g., admin API, Mailpit, Grafana)
- **Lab:** you will use OpenFeature and Unleash
  - these will be used for your experiment in Checkpoint B

🎯 Learning Goals

- Distinguish technical risk from product risk in a proposed change, and explain how separating deployment from release helps limit both.
- Choose an exposure strategy for a change based on its risks, its reversibility, and the evidence needed.
- Turn a product idea into a testable hypothesis with an outcome metric, guardrails, and a decision rule, and identify a cheaper test that could disprove it.
- Explain why randomized comparison is needed to attribute an outcome to a change, and diagnose why a claimed result might be misleading.

Technical and Product Risk

Why is shipping a change risky even when the code works?

Vote: Which of these changes helped?

1 **MSN home page: open Hotmail in a new tab instead of the same tab.**

HELPED — +8.9% engagement; replicated with 2.7 million US users.

2 **Bing: add a pane of Facebook and Twitter content to search results.**

NO EFFECT — Over $25 million to build; negligible engagement or revenue.

3 **Bing: make title blues and greens slightly darker.**

HELPED — Faster, more successful searches; over $10 million a year.

4 **Bing: show fewer ads, but make each one larger.**

HELPED — Over $50 million a year without hurting user experience.

Kohavi and Thomke, “The Surprising Power of Online Experiments,” Harvard Business Review, 2017.

A shelved idea earned over $100M a year

+12%

Bing revenue from longer ad titles

Low priority	The idea sat in the backlog for more than six months.
Cheap to test	An engineer ran an A/B test because the code took days to write.
Usually wrong	At Google and Bing, only about 10–20% of experiments produce positive results.

Untested judgment builds losing ideas and shelves winning ones.

> “The most inefficient way to test a business model or product idea is to build the complete product to see whether the predicted demand actually exists.”



Jez Humble
Coauthor of The DevOps Handbook
Quoted in The DevOps Handbook



THE WIRED APP IS HERE | DOWNLOAD NOW »

WIRED SECURITY POLITICS THE BIG STORY BUSINESS SCIENCE CULTURE REVIEWS NEWSLETTERS

FEATURED LAST MINUTE PRIME DAY DEALS BEST SMART SCALES BEST KINDLES BEST MAC DESKTOP DEALS DELIVERED GIFT GUIDES

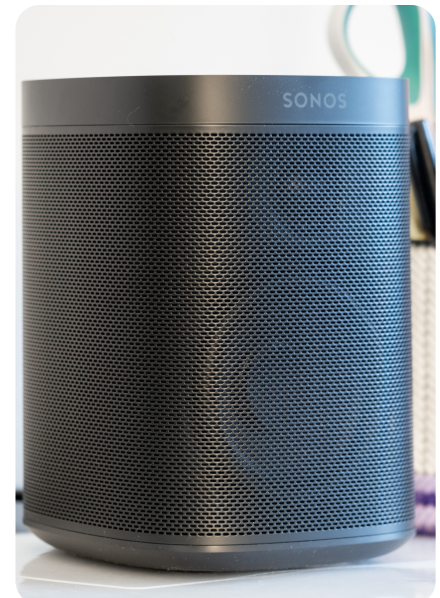
BOONE ASHWORTH GEAR MAY 11, 2024 7:30 AM

Many People Do Not Like the New Sonos App

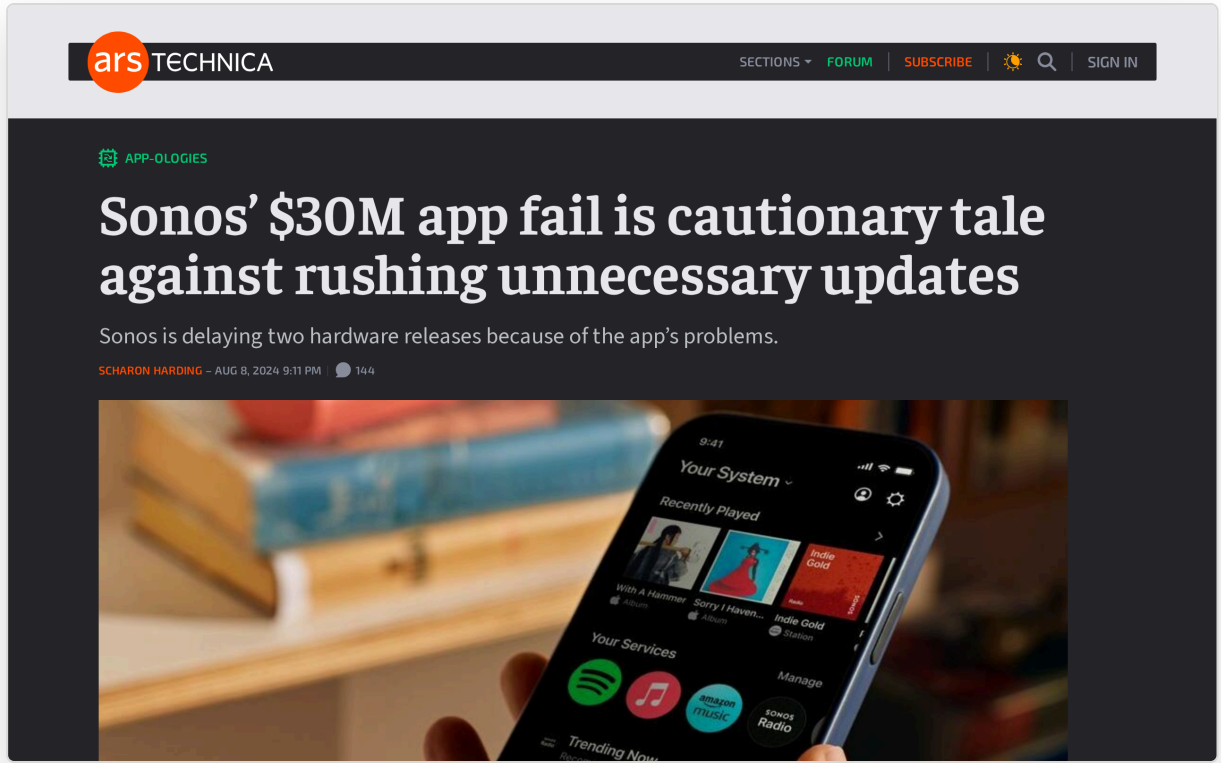
Plus: Apple crushes artistic dreams, Comcast introduces low-cost internet plans, and Black Twitter will always be Black Twitter.



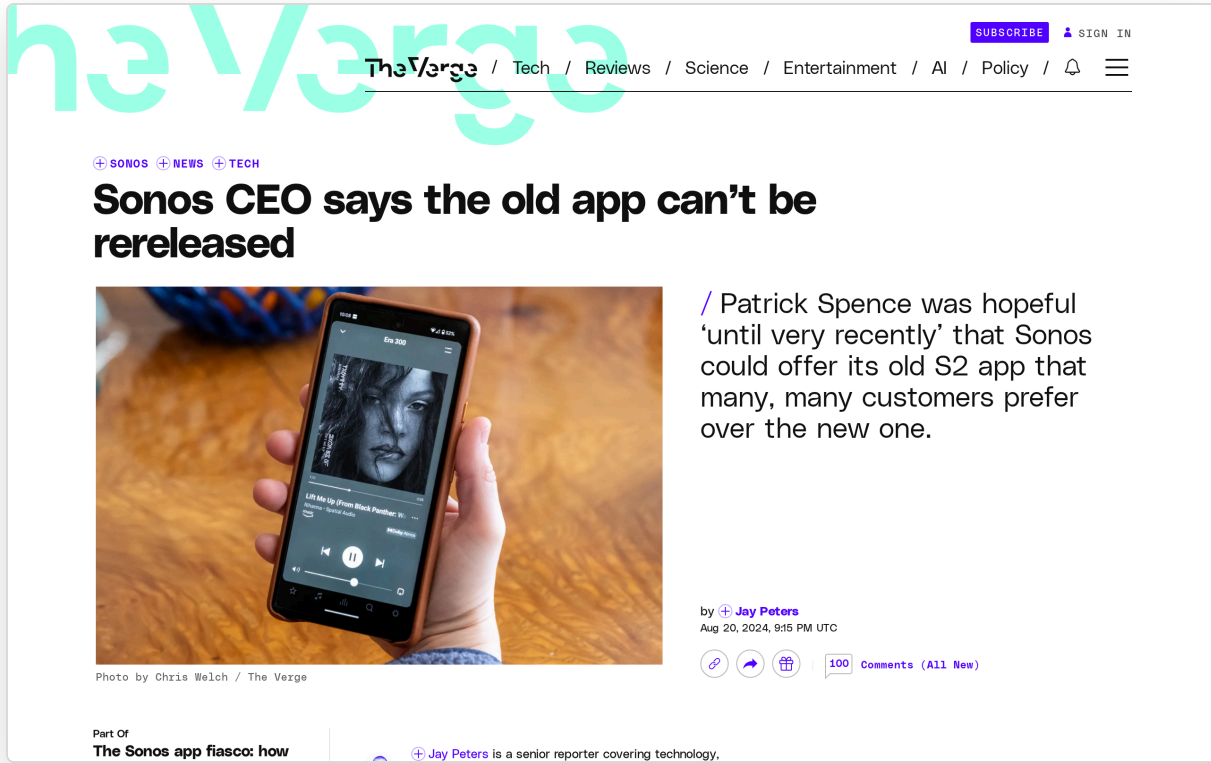
WIRED · May 11, 2024



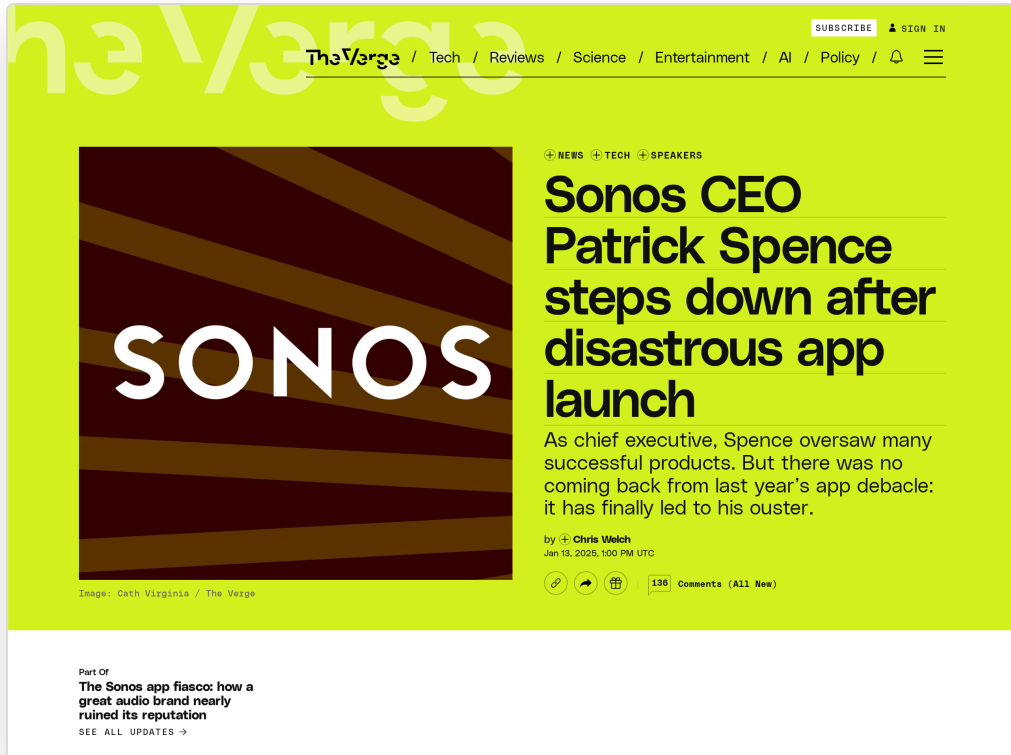
Sonos One · Ajay Suresh · CC BY 2.0



Ars Technica · August 8, 2024



The Verge · August 20, 2024



The Verge · January 13, 2025

Sonos: a broad release with no easy rollback

- **The system:** a phone app controls speakers that run firmware and use cloud services.
- **May 2024:** the rewrite replaced the S2 app, removing familiar features and introducing reliability problems.
- **August 2024:** Sonos said firmware and cloud changes prevented restoring the old S2 app.



A broad release caused harm that an app rollback could not undo.

A release can fail by breaking or by not helping

Technical risk

The change harms the service: errors, latency, crashes, or lost data.

Evidence: health signals from the exposed users.

Product risk

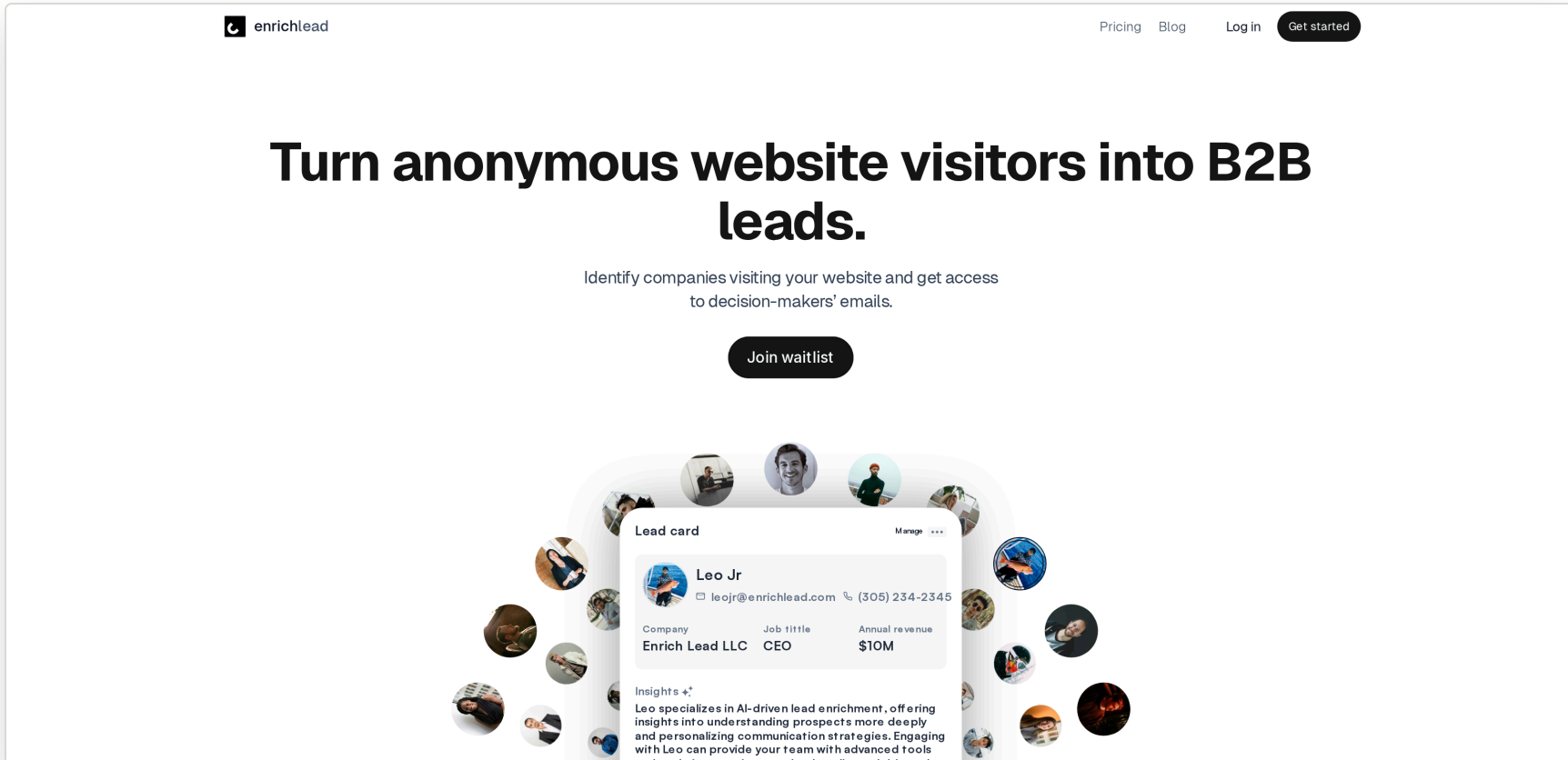
The change works as built but users are no better off, or worse.

Evidence: a user outcome compared with users who did not get the change.

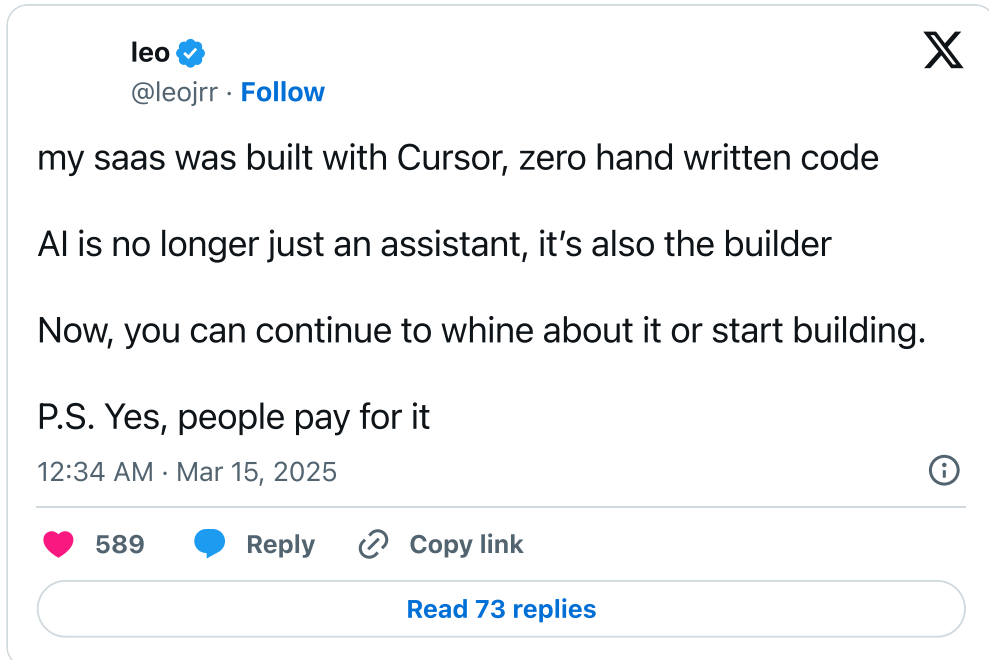
Which risks did Sonos face? Which did Bing's longer ad titles face?

- Sonos suffered both.
- Bing's longer ad titles carried almost no technical risk but real product uncertainty.

Risk grows when you ship code you do not understand



Enrichlead · original marketing homepage



leo
@leojrr · Follow

my saas was built with Cursor, zero hand written code

AI is no longer just an assistant, it's also the builder

Now, you can continue to whine about it or start building.

P.S. Yes, people pay for it

12:34 AM · Mar 15, 2025

589 Reply Copy link

Read 73 replies



leo
@leojrr · Follow

guys, i'm under attack

ever since I started to share how I built my SaaS using Cursor

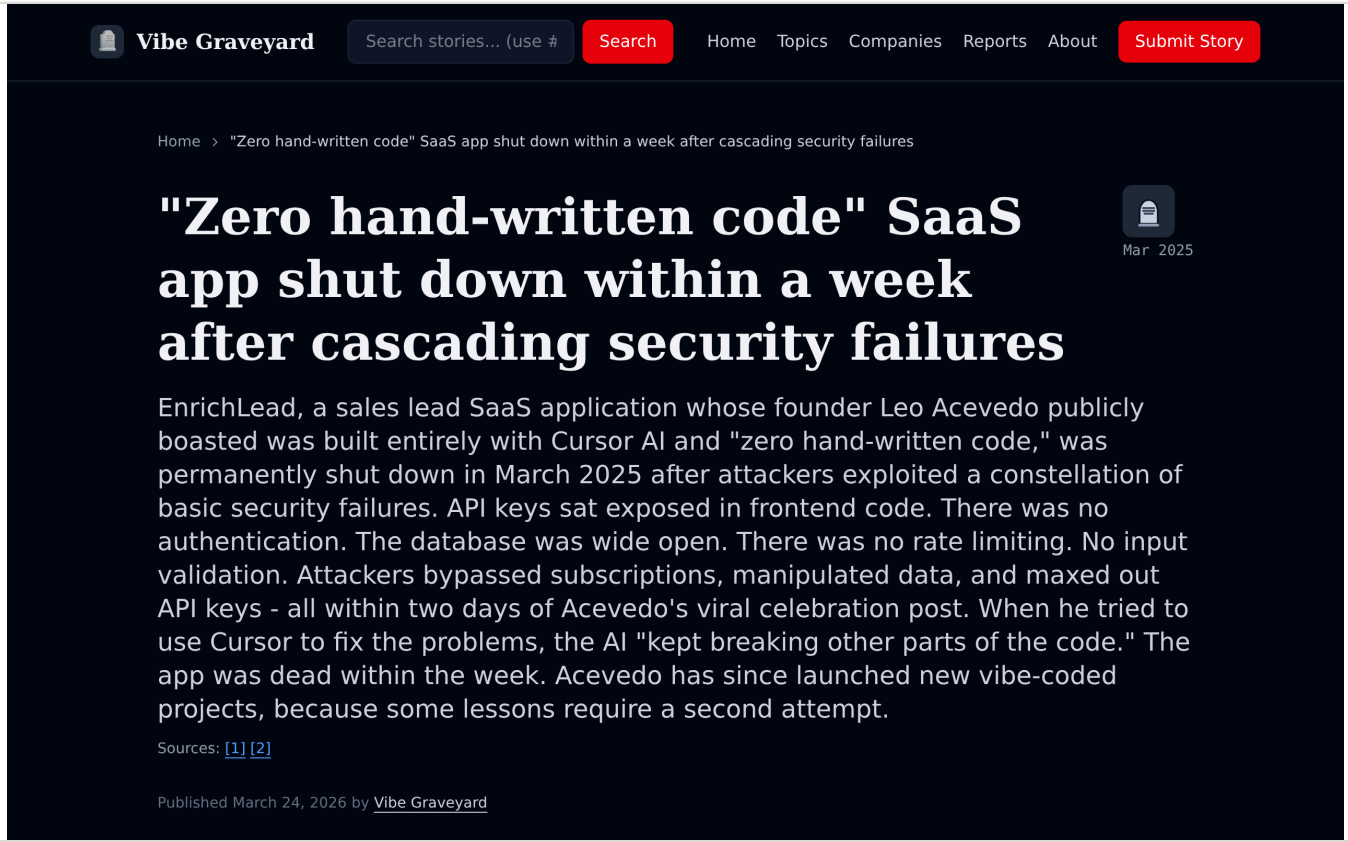
random thing are happening, maxed out usage on api keys, people bypassing the subscription, creating random shit on db

as you know, I'm not technical so this is taking me longer that usual to Show more

5:04 AM · Mar 17, 2025

6K Reply Copy link

Read 602 replies



Vibe Graveyard · March 24, 2026

Today: How do we control risk?

Limit exposure and test whether changes help



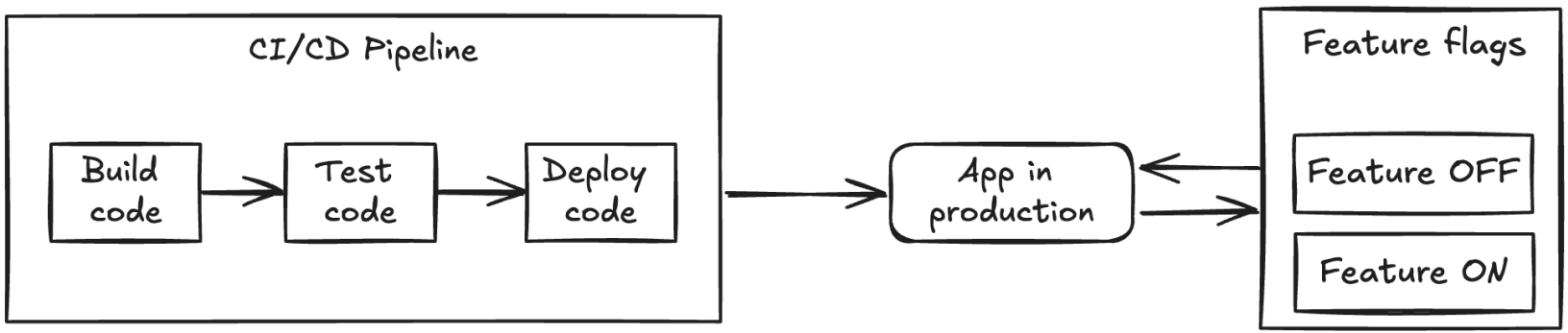
Exposure control
Limits how many users a bad change can reach, whichever way it fails.



Experiments
Show whether the change actually improved what users achieve.

Controlling Exposure

How can we limit who encounters a change?



Unleash | Apache 2.0

A feature flag selects behavior at runtime



KEY CONCEPT

Feature flag

A configured choice between code paths, evaluated for each user at runtime. Allows us to switch behavior without redeploying.

```
if flags.enabled("new-ranking", user):
    results = new_ranker.rank(query)
else:
    results = old_ranker.rank(query)
```

Flags can select a value at runtime

OpenFeature provides a standard API for evaluating flags. A **provider** connects it to your flag system.

```python
from openfeature import api
from openfeature.evaluation_context import EvaluationContext

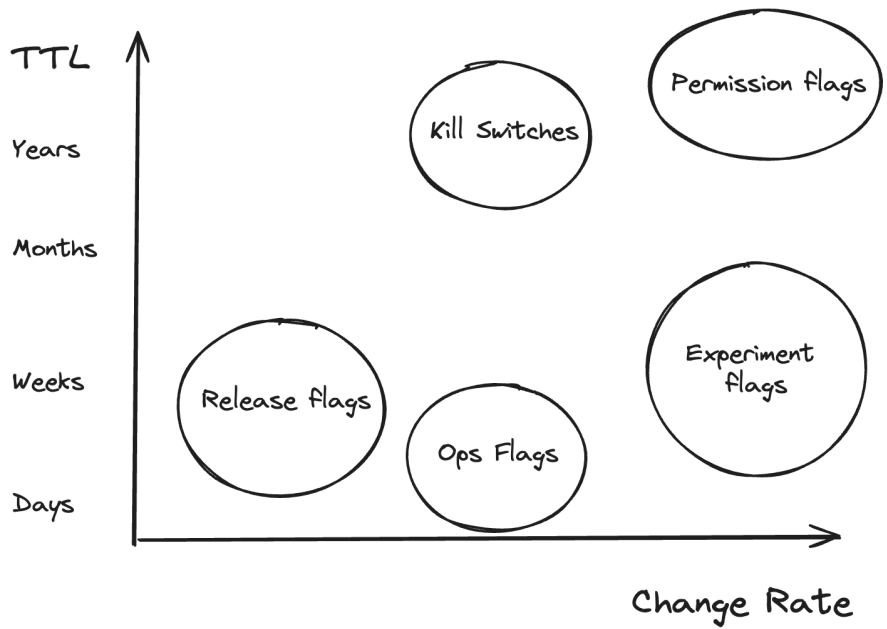
client = api.get_client()
context = EvaluationContext(targeting_key=user.id)

enabled = client.get_boolean_value("new-ranking", False, context)
ranker = client.get_string_value("ranking-method", "popular", context)
limit = client.get_integer_value("result-limit", 20, context)
```

The second argument is the **default value**: use it when the flag is missing or evaluation fails.

OpenFeature · Python SDK

Flags serve different purposes



- **Release:** roll out new behavior
- **Experiment:** compare variants
- **Ops:** temporarily enable tracing
- **Kill switch:** stop failing behavior
- **Permission:** select by plan or role

TTL = flag lifetime. Unleash · Feature flag types · taxonomy credited to Pete Hodgson

Many tools manage feature flags













We will use **Unleash** for the examples that follow.

Progressive rollout widens exposure stage by stage



Canary
Evaluate, then wait

Wider
Evaluate, then wait

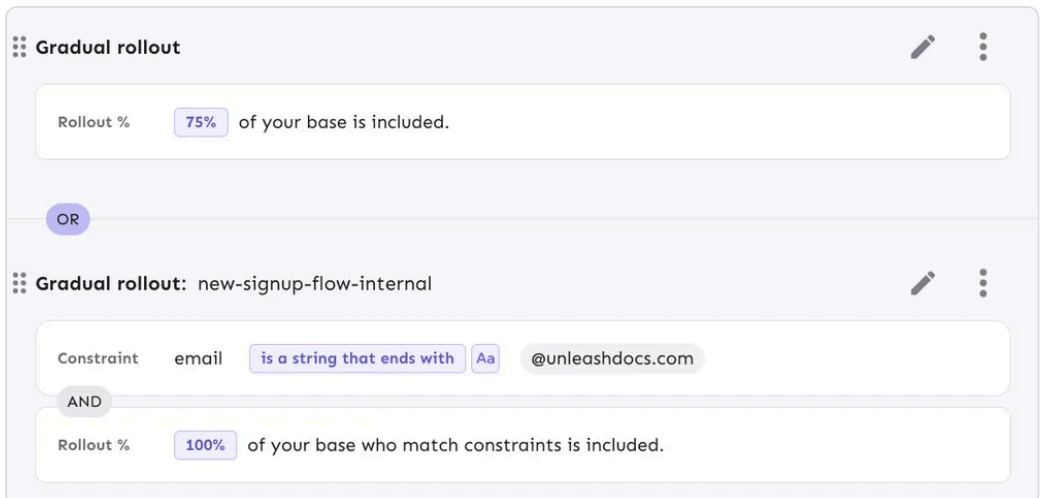
Most users
Evaluate, then wait

Everyone
Full release

Recall: CrowdStrike, July 2024

A faulty content update reached every Windows host running the sensor at once, and the hosts crashed. The company’s root-cause report adds staged rings, canaries, and bake time.

Strategies decide who receives the feature



Example: All employees + 75% of other users

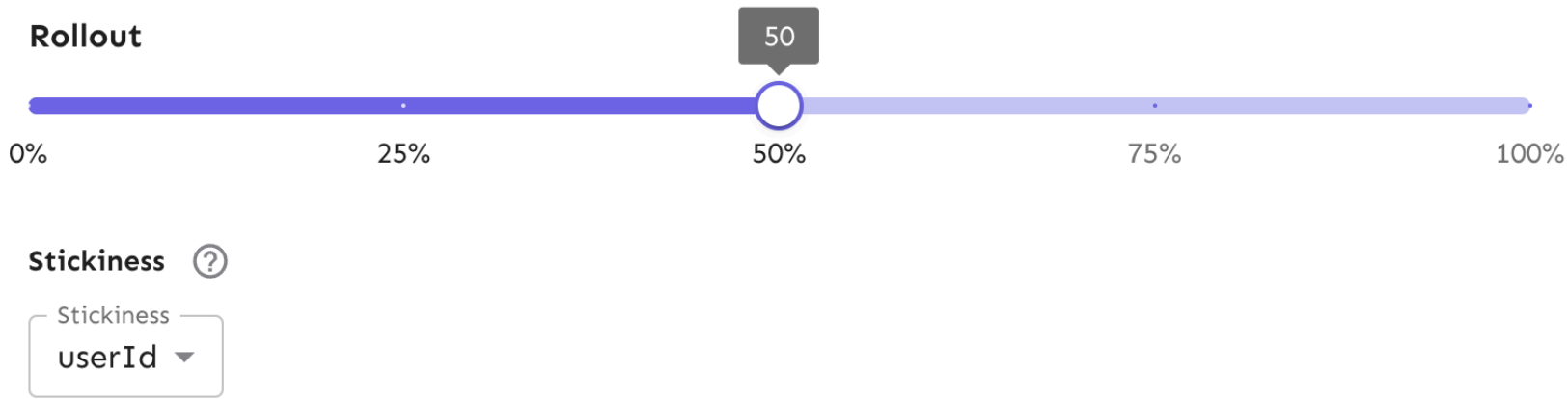
Unleash · Activation strategies

Choose a **rollout percentage** and **targeting rules**.

AND within a strategy: all constraints must match.

OR between strategies: any one can enable the flag.

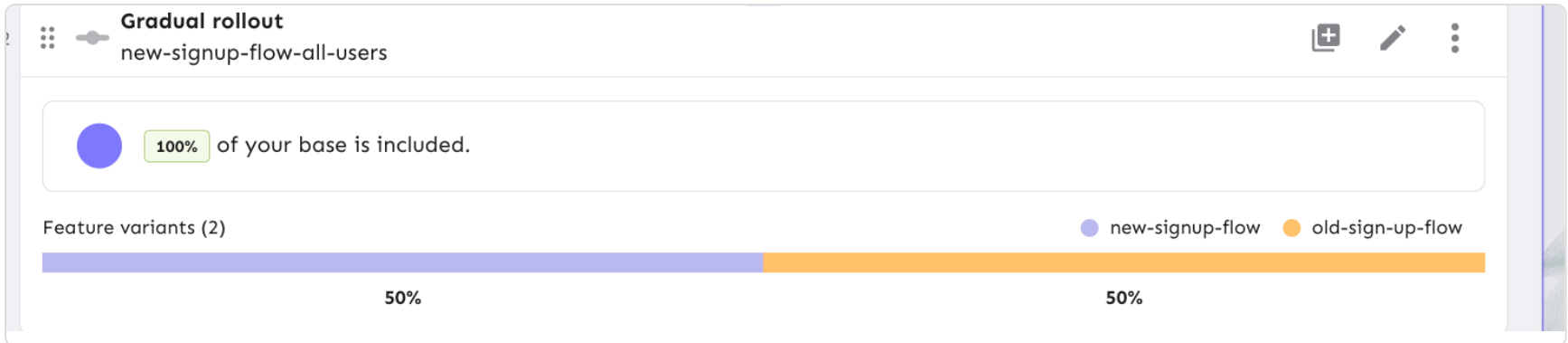
Roll out to a stable group of users



- **Rollout:** choose a percentage of eligible users
- **Stickiness:** use a consistent identity to preserve assignments
- **Expansion:** add users while keeping the current group included

Unleash · Rollout · Stickiness · v6.9.0 interface

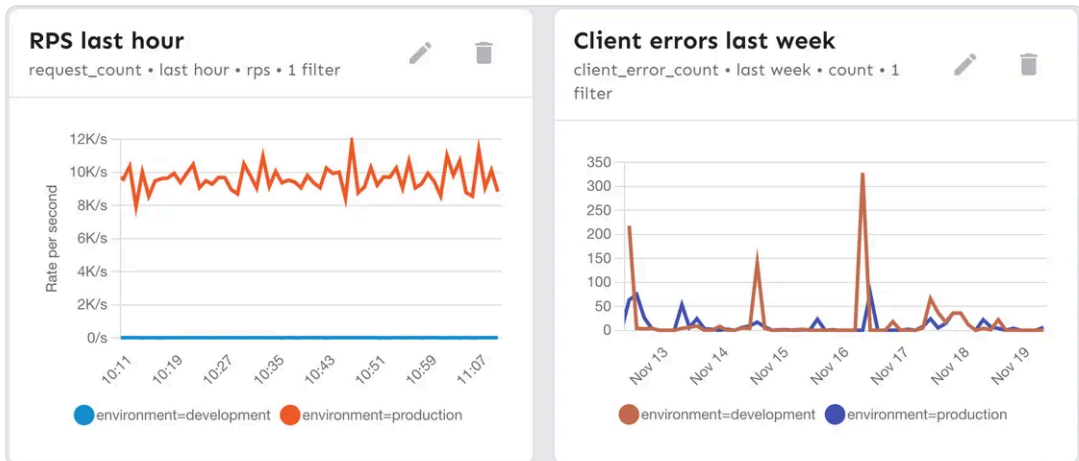
Variants select which behavior users receive



- **Variants:** name the alternative behaviors
- **Weights:** divide enabled users between them

Unleash · Strategy variants · v6.9.0 interface

Capture metrics before you start the rollout



- **Record:** capture errors, latency, and relevant user outcomes
- **Connect:** let Unleash query Prometheus or VictoriaMetrics
- **Verify:** check that data arrives before expanding exposure

Unleash · Connect external metrics

Use those metrics to enforce stop criteria



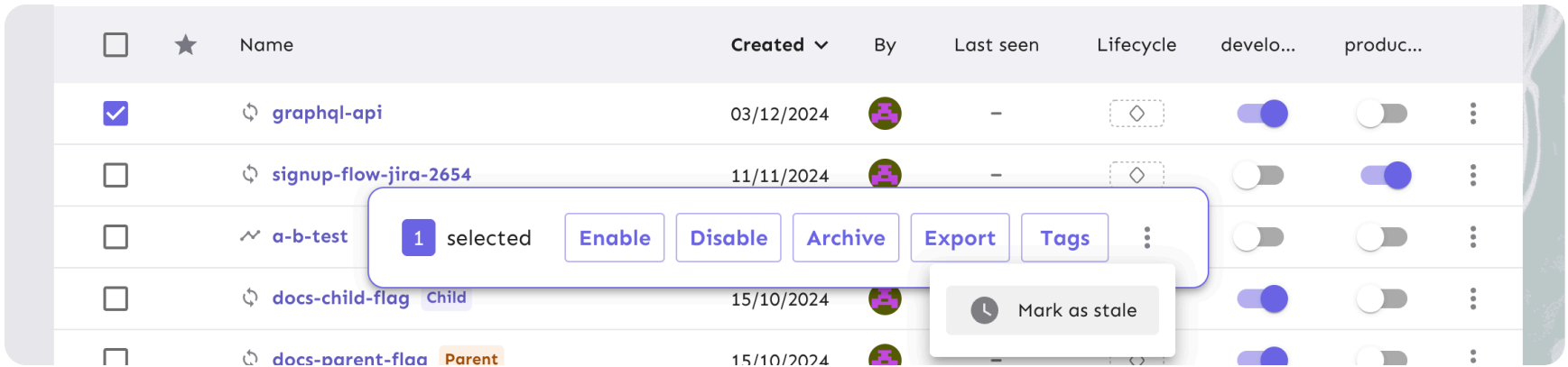
Example rule

Error rate exceeds 1% for 5 minutes
→ Disable the flag.

- **Pause:** keep current exposure while you investigate
- **Disable:** switch to the fallback when a limit is exceeded

Unleash · Metric safeguards

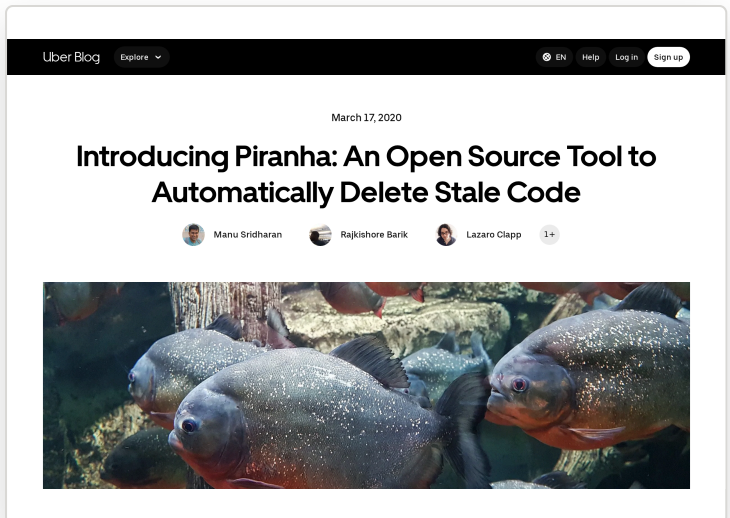
Inventory flags and retire the ones you no longer need



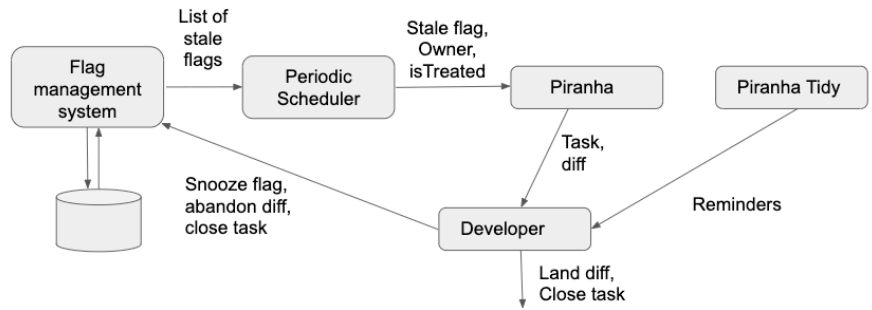
- **Inventory:** track each flag's owner, purpose, and expected lifetime
- **Review:** identify flags that have outlived their purpose
- **Retire:** remove unused checks and code paths, then archive

Unleash · Flag inventory and stale flags

Uber automated stale-flag cleanup; owners still decide



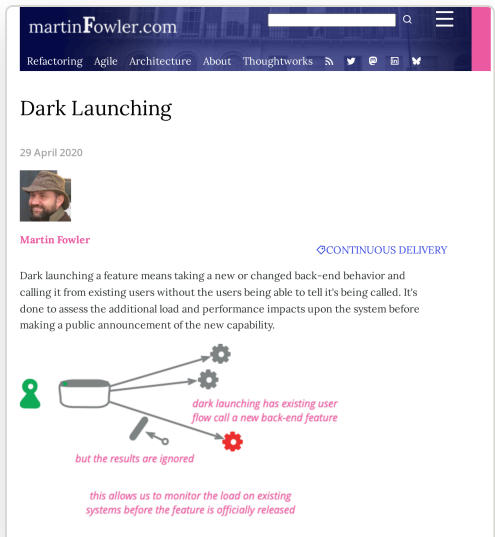
Uber Blog · March 17, 2020



- **Find:** flags left unchanged for weeks are treated as stale
- **Generate:** Piranha deletes the flag check, the dead path, and its tests
- **Review:** the flag's owner lands the change or marks the flag as still needed

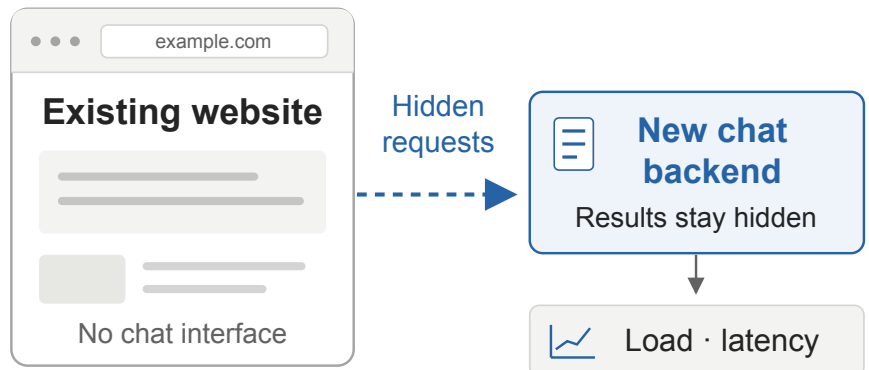
Can we test a feature without showing it to users?

A dark launch runs behavior before users see it



Martin Fowler · April 29, 2020

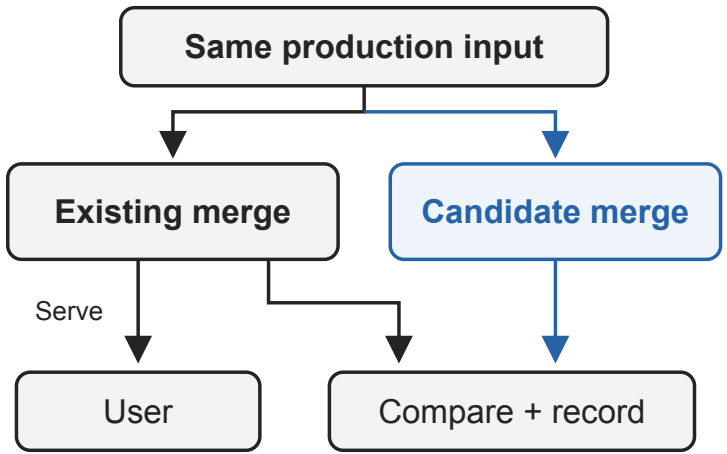
Facebook Chat: browsers sent hidden traffic before the chat interface appeared.



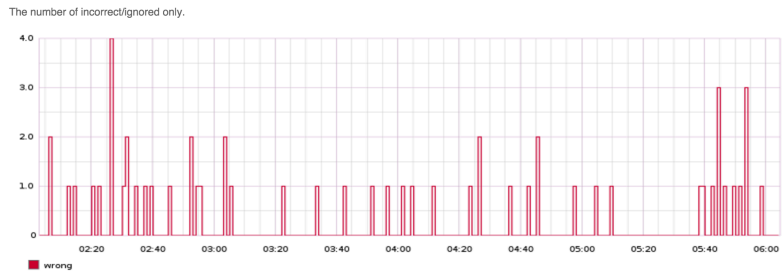
Can the backend handle real traffic? Hidden work still uses **production capacity**.

The DevOps Handbook · Chapter 12 · Fowler · Dark Launching

Shadow execution compares code on real inputs



GitHub was replacing its merge implementation. **Scientist** ran both paths on selected production requests and returned the existing result.



Record differences and execution times. Mismatches exposed bugs in **both** implementations.

GitHub · Move Fast and Fix Things (2015)

Each exposure control answers a different question

Control	Selects	Evidence about	Cost
Feature flag	Behavior, per user	Whichever risk you measure	Code paths to maintain and remove
Progressive rollout	How many users	Technical risk, stage by stage	Slower full release
Dark launch / shadow	Hidden execution	Technical risk only	Extra load; side effects to isolate

All of these controls can be combined.

⚠ Not all changes are easily reversible

Shared dependencies

A 1% canary that overloads a shared cache harms every user.

Side effects

A shadow request that charges a card or sends an email is real.

Code on devices

Shipped app code persists; keep its behavior controllable from the server.

Broken machines and data

A crashed host may be unreachable; migrated data stays migrated.

Put the most caution where effects cannot be reversed.

Activity: How Would You Ship This?

What exposure does this change deserve?

Decide how one change should reach users



1 · What could go wrong?

Is the risk technical, product, or both?



2 · How will you roll it out?

How you control who sees it, who sees it first, and how exposure grows.



3 · When will you stop?

The evidence that would make you pause or reverse.



4 · What can't you take back?

What turning the change off would not undo.

Each team gets one scenario card.

The dominant constraint shapes each plan

1 Switch payment providers

SIDE EFFECTS — Shadowing a charge charges the customer twice.

2 New search-ranking model

BOTH RISKS — Shadow shows it is safe, not that results are better.

3 Mobile app redesign

SHIPPED CODE — Phones keep old code; control release from the server.

4 Raise free-shipping threshold

PRODUCT ONLY — One-line change; the question is all evidence.

5 Config push to customer machines

UNREACHABLE — A crashed machine cannot fetch the fix.

6 Replace the cache client

SHARED DEPENDENCY — A small canary can still harm everyone.

A stop signal must match the change's risk

Catches technical failure

Error rate, latency, crash reports, failed payments, compared with the existing version.

Catches product failure

The user outcome the change was meant to improve: orders completed, searches that end in a click and a stay.

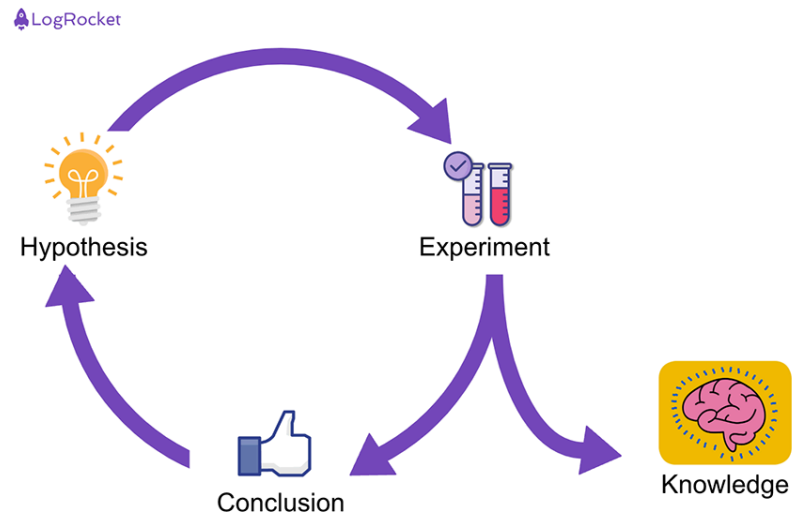
A product-risk change whose plan stops only on errors **will never stop**.

Hypothesis-Driven Development

How do we turn an idea into a claim we can disprove?

Hypothesis-driven development tests predictions about user behavior

- State the change you expect, test it, and use the evidence to decide what to build next.
- The result can support the prediction or give you a reason to **revise or stop**.



Bart Krawczyk, LogRocket, 2024.

State the change, the users, and the expected result

Product hypothesis — Delivery dates

Change	Show delivery dates on product pages
Users	First-time visitors
Expected outcome	More visitors complete an order
Measure	Percentage who order within 7 days of their first visit
Minimum worthwhile gain	+1 percentage point compared with the existing page

Choose these before collecting results.

A payment app wants a “split the bill” button. What is the cheapest test that could prove it unwanted?



Disprove ideas before writing production code

Fake door

A button for a feature that does not exist yet; count the clicks.

Prototype

A clickable mockup tried by a handful of real users.

Wizard of Oz

People do the work by hand behind an automated-looking interface.

Zappos, 1999

Posted photos of shoes in local stores; bought each pair only after a customer ordered it.

Each tests **demand**, not implementation.

Measure whether users reached their goal



Browse
Sees the home page

Click a title
Proxy metric

Start playing
Proxy metric

Keep watching
Outcome: found something

- A change can raise clicks while fewer people find something to watch. Proxies help diagnose the journey; the **outcome** decides
- **Example:** Duolingo judged faster app start-up by whether learners went on to complete lessons, not by latency alone

Guardrail metrics set limits on acceptable harm

A **guardrail** is a limit on harm while you improve the main outcome.

Metric to protect	Maximum allowed increase vs. existing behavior
Checkout error rate	**0.2** percentage points
p95 page latency	**100** ms
Unsubscribe rate	**0** no increase

- If a guardrail breaks, **pause expansion** (even if orders increase)

Choose the smallest gain worth building for

- **Worthwhile effect:** the smallest improvement worth the cost of building and maintaining the change.
- **Choose it before collecting results.** A smaller gain may not justify the cost of building and maintaining the feature.

First-time visitors who order within 7 days

10% + **1 pp** = **11%**

existing | worthwhile | target

Randomized Experiments

How can we tell whether the change caused the outcome?

Did Yahoo's ads cause more brand searches?

Estimated increase in the share of users searching for the brand:

871–1,198%
Ad viewers vs. non-viewers with similar past activity

5.4%
Randomly assigned: brand ad vs. unrelated ad



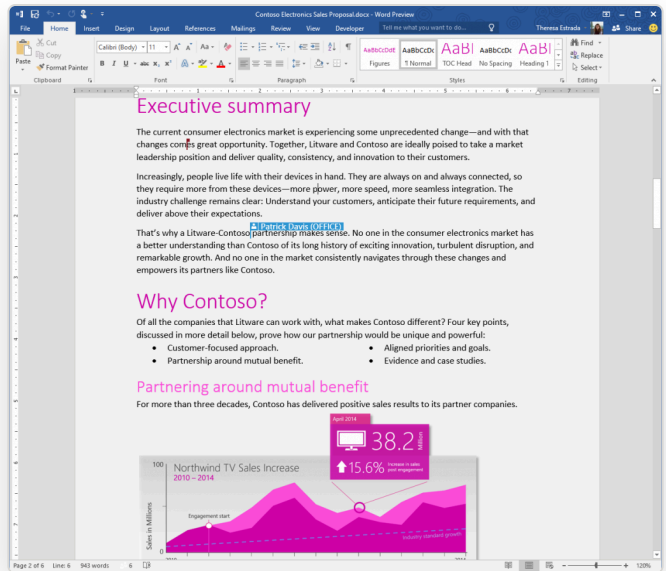
Coolcaesar / Wikimedia Commons, CC BY-SA 3.0.

- More active users both saw more ads and searched more.
- Random assignment balanced activity across the groups.

Office users who tried new features stayed longer

- Users who tried **advanced features** were more likely to keep using Office
- Users who saw **error messages** were also more likely to keep using Office

Should we add more advanced features—and show more error messages—to keep users?



Case: Kohavi and Thomke, HBR 2017.

Selection bias



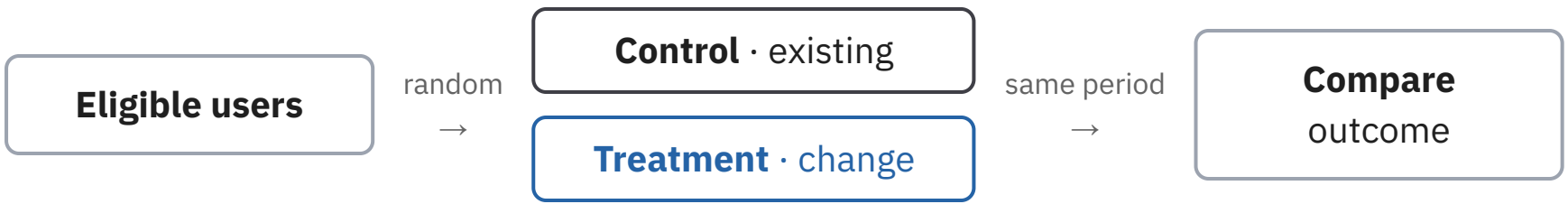
KEY CONCEPT

Selection bias

The groups being compared differed before the change, so their outcomes would differ anyway.

- Frequent users encounter more features and more errors—and are more likely to keep using Office.

Random assignment makes the groups comparable



- Other influences balance out on average, so the difference **estimates the effect**.
- Chance still leaves an **uncertainty range** that narrows with more users. Fix the duration in advance to reach enough users and cover full weekly cycles.

Each user keeps one variant for the whole test



Per request

A shopper sees the old page, then the new one, then the old one.

Which variant caused the purchase?

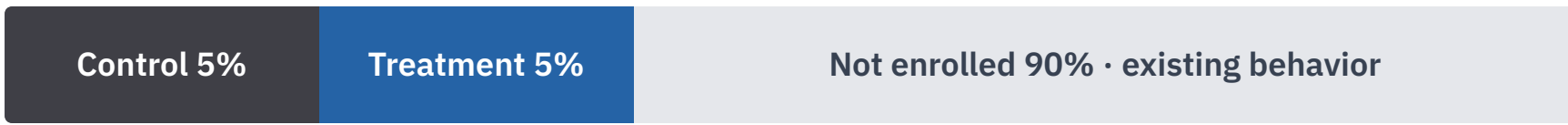


Per user

Each account is assigned once and keeps its variant on every request and return visit.

Here, the **unit of assignment** is the user.

Experiments can run inside a limited rollout



- A flag enrolls 10% of eligible users and splits them evenly.
- Everyone else keeps the existing behavior.
- Conclusions apply only to the **population and conditions tested**.

Activity: Design an Experiment

Would this experiment tell us the truth?

Plan a test that could prove your idea wrong

Your group's idea

- **A · One-tap reorder**

  Grocery app: refill last week's basket

- **B · 30-day free trial**

  Streaming service: up from 14 days

- **C · "Only 3 left"**

  Online store: shown on low-stock items

Experiment card

1. What do you predict?
2. Who takes part?
3. How will you know?
4. What will you do with the result?

Swap cards: how could this result fool you?

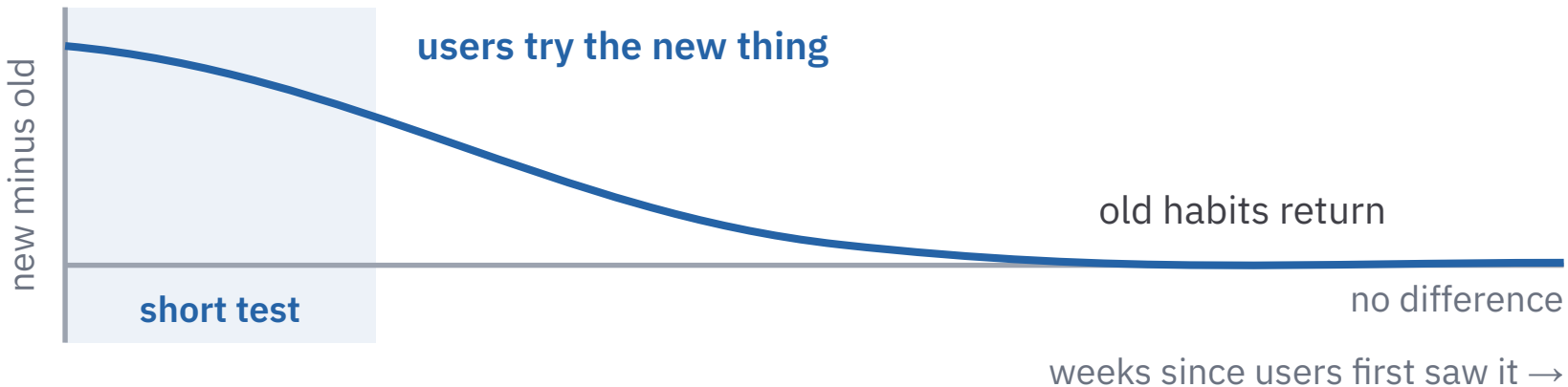


⚠ Stopping at a lucky peak manufactures a win



- Don’t claim a win until the test has run **as long as you planned**.
- You can still stop early if a number you’re protecting gets worse (guardrail metric).

⚠ An early boost from novelty may not last



- **Novelty effect:** users try something because it's new, then drift back to old habits
- Test for long enough for the novelty to **fade**: a short test can't show a lasting gain

⚠ Users missing from one group can fake a win



- Only patient users were counted, so the new page looks **better** than it is.
- Before reading the result, **check that group sizes match the plan**.

Acting on the Result

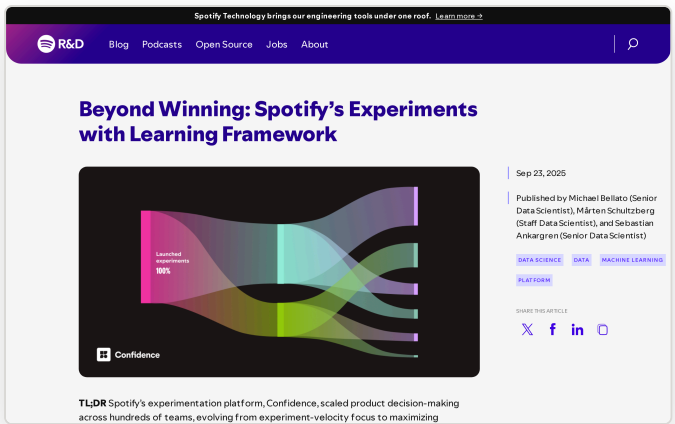
What do we do with the evidence and the flag?

Let the evidence choose the release action

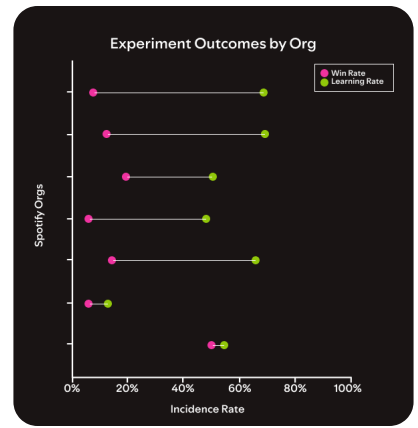
Evidence	Action
Worthwhile gain, guardrails hold	**Expand**, and keep monitoring
Gain, but a guardrail breaks	**Hold**, mitigate, revise
Precise effect below the threshold	**Revise or abandon**
Inconclusive: range spans zero and the target	Keep exposure limited; **measure more** or stop
Broken assignment or logging	**Fix the design** and rerun

An inconclusive result is **not** evidence of no effect.

Use negative results to inform the next experiment



Spotify Engineering · September 23, 2025



- About **64%** of Spotify's experiments produced learning; only about **12%** were wins.
- Learning means a win, a caught regression, or clear evidence that the effect is too small to matter.

TL;DR: Limit exposure and test whether changes help

Technical risk
It breaks

Flags, kill switches, progressive rollout, dark launches, and stop criteria set in advance.

Product risk
It does not help

A written hypothesis, cheap tests first, then a randomized comparison on a user outcome with guardrails.

Sonos needed both. The Bing ad titles needed only a cheap test.

🚀 Today, we learned how to ...

- Distinguish technical risk from product risk in a proposed change, and explain how separating deployment from release helps limit both.
- Choose an exposure strategy for a change based on its risks, its reversibility, and the evidence needed.
- Turn a product idea into a testable hypothesis with an outcome metric, guardrails, and a decision rule, and identify a cheaper test that could disprove it.
- Explain why randomized comparison is needed to attribute an outcome to a change, and diagnose why a claimed result might be misleading.