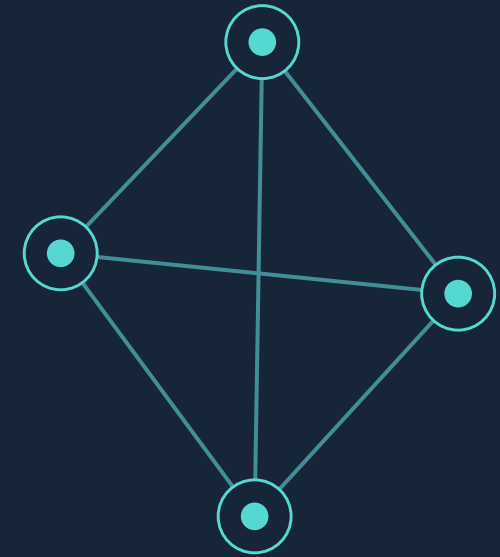


Stability Is a **System**, Not a Metric

Lessons from Scaling Mobile
Test Automation

Pranav Pandit

Staff Quality Engineer | Robinhood



**One connected
system.**

From maintenance pressure to useful automation

The problem

Recurring failures took time away from feature testing.



What we changed

Failure handling, priorities, infrastructure and AI-assisted investigation.



What we learned

What improved, what didn't, and what still needed work.

THE PROBLEM

A bigger suite. A busier QE team.

January 2026 | mobile automation across regional apps

SCALE

~2,000

Appium tests per platform

RUN CADENCE

**Nightly
+ beta**

Regular runs and release checks

QE WORK

**Feature testing
+ maintenance**

The same team supported both

We struggled to trust the results

January 2026 | US app | all tests

iOS

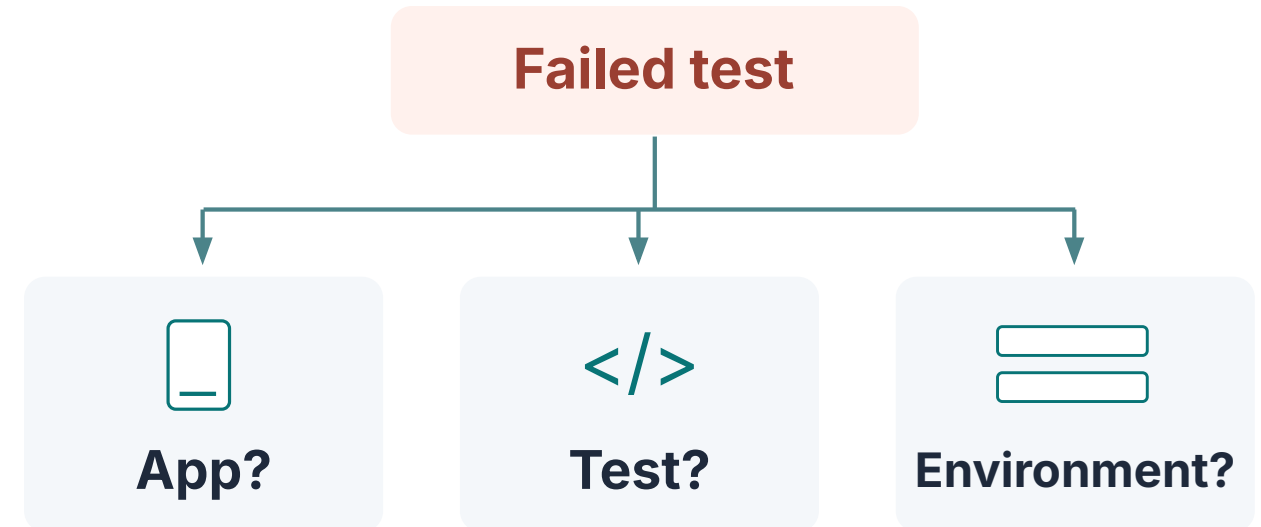
Android

59.0% **48.5%**

Reported average pass rates

All tests; priority tagging
was not yet in place.

What caused the failure?



Source: internal January 2026 summary

Unstable automation takes QE time

New-feature testing was our core work.

Recurring failures

Investigate, rerun
and repair



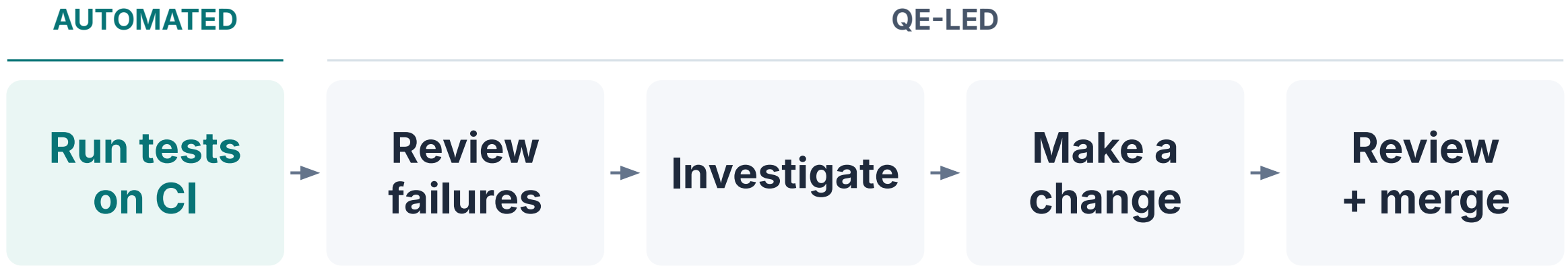
Unresolved failures

Manually recheck
existing journeys



Less time for new-feature testing

The run was automatic. The follow-up wasn't.



Repeated investigation. Repairs needed follow-up.

Four themes guided our improvements

Failure handling

Ownership and recovery checks

Prioritisation

Focus on critical customer journeys

Infrastructure

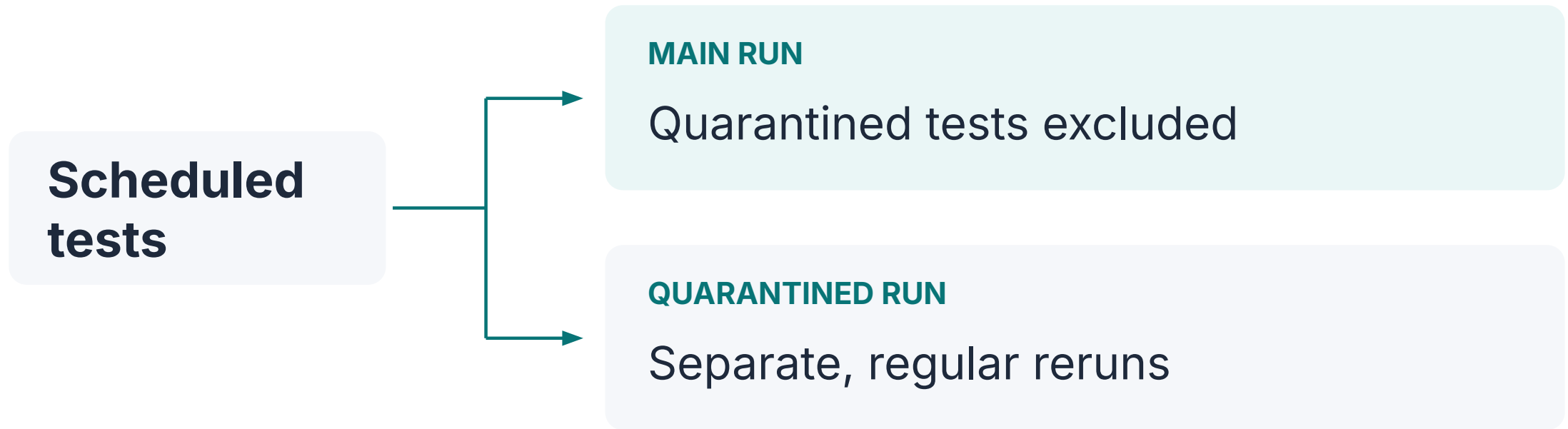
Shared services and test environments

Scale the response

AI-assisted investigation and repair

We ran recurring failures separately

Early quarantine approach



Separate runs. Clear owners. A checked route back.

We tracked the backlog alongside pass rate

Pass rate

How did the tests that ran perform?

Quarantine backlog

Which tests were still there, and for how long?

Restore useful checks, not just a better-looking report.

QEs were fixing tests. Some still came back.

Better organised work

Fixes made and tickets closed

Tests Came back

Some passed locally but failed in CI

Time was still limited

Repairs competed with feature testing

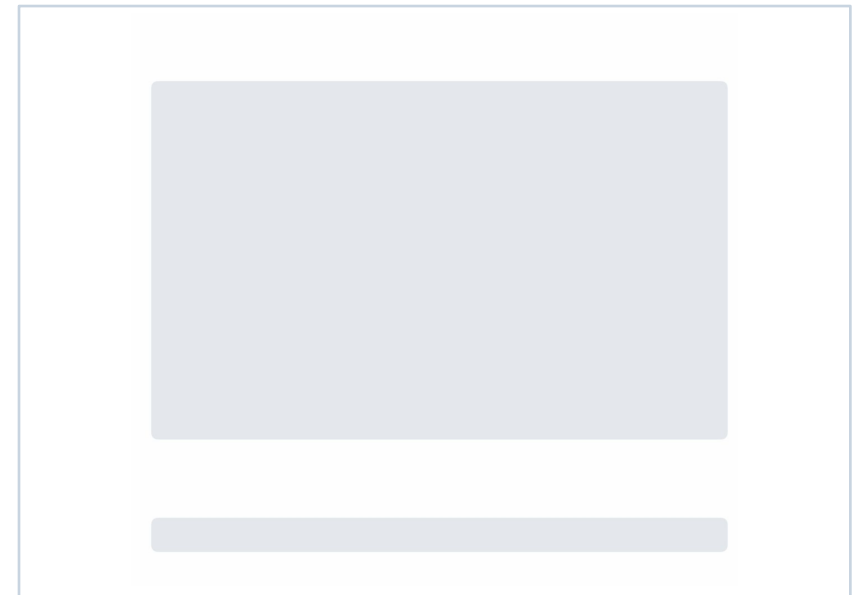
"Element not found" was not a diagnosis

A real error and its captured screen

REPORTED ERROR

```
NoSuchElementException  
Element not found
```

CAPTURED SCREEN (CROPPED)



Loading placeholders visible

We found common problems behind different failures

Setup failed silently

Expected test data was never created

Teardown failed

Next test would start with wrong state

Parallel tests interfered

One test changed another test's data

Look at the shared steps, not just the final error.

We focused repairs on critical journeys

P0 = mission-critical | P1 = critical

Critical journeys

Smaller set to repair first

Quarantined tests still ran
and counted in the core nightly run.

Wider coverage

Separate reporting and priorities

Keep gaps, overlap and
obsolete tests visible.

Prioritisation made the next step clearer

What helped

A focused starting point for repairs

What still needed work

Keeping mappings and wider coverage current

Critical tests still depended on the same environment.

Why did passing tests fail in larger runs?

Earlier beta investigation | May 2026

What we noticed

Some tests passed nightly but failed in the larger beta run.

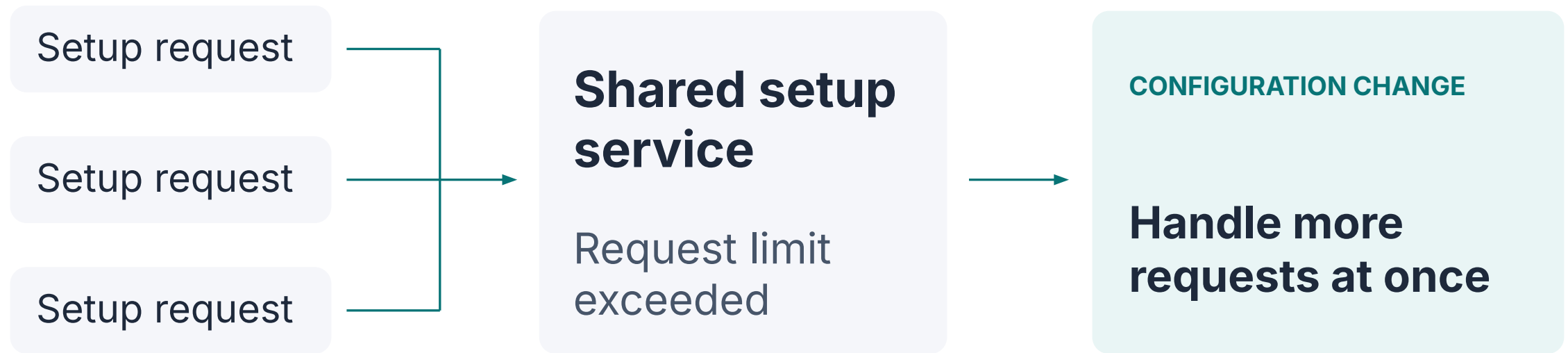
What we investigated

Were parallel setup requests putting pressure on shared services?

Look at how the tests run together, not just which tests failed.

Parallel setup hit a shared-service limit

SEVERAL TESTS PREPARING DATA



The change addressed this shared setup bottleneck.

AI helped us find the right next step

mobile-debug AI agent

Run evidence + test code + app context



Test-code issue

Propose a repair
Validate on a real device

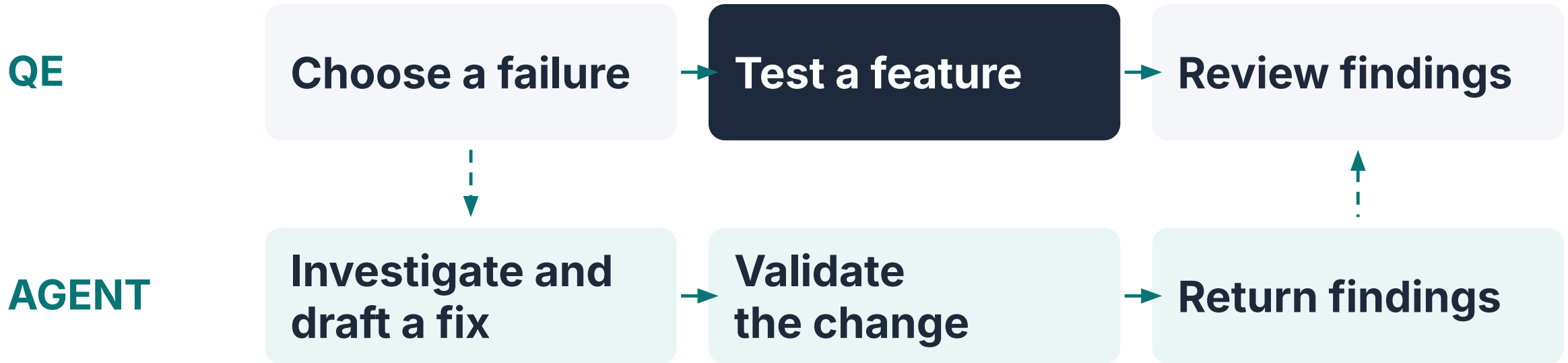
Environment issue

Return findings
Involve the service owner

Product defect or unclear cause? Return findings for investigation.

Investigation ran alongside feature testing

The mobile-debug AI agent could work in the background.



LOCAL

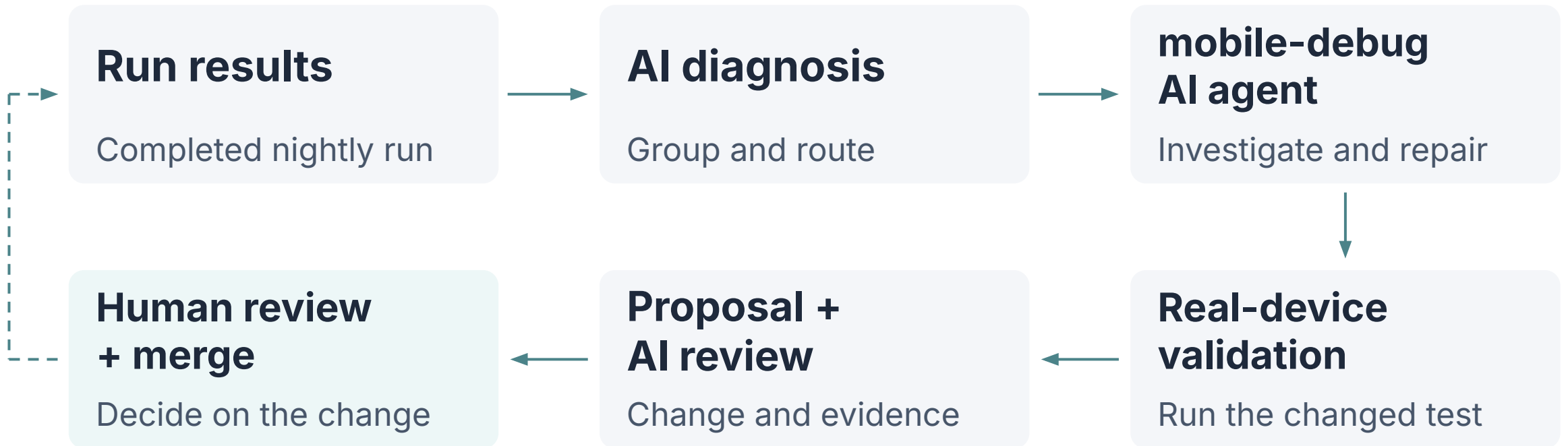
QE-guided investigation

SANDBOXED

Work in a separate environment

We connected the steps, with human review

Repair path | September 2026



Check later runs too.

We set limits on automated repairs

Verify the cause

Check app source,
not just a screenshot.

Stop repeated guessing

No new evidence?
Return the findings.

Check shared changes

Look for conflicting
edits to shared files.

A passing test must still check the right behaviour.

The next investigation started with the history

TEST HISTORY

What failed

What we tried

What ran

What happened next

Including unsuccessful attempts



Next investigation

Start with what we know

Passing once did not prove the fix would last

Review of test-fix outcomes

Kept passing

No further failures
observed

Passed again later

Failed after the fix;
latest run passed

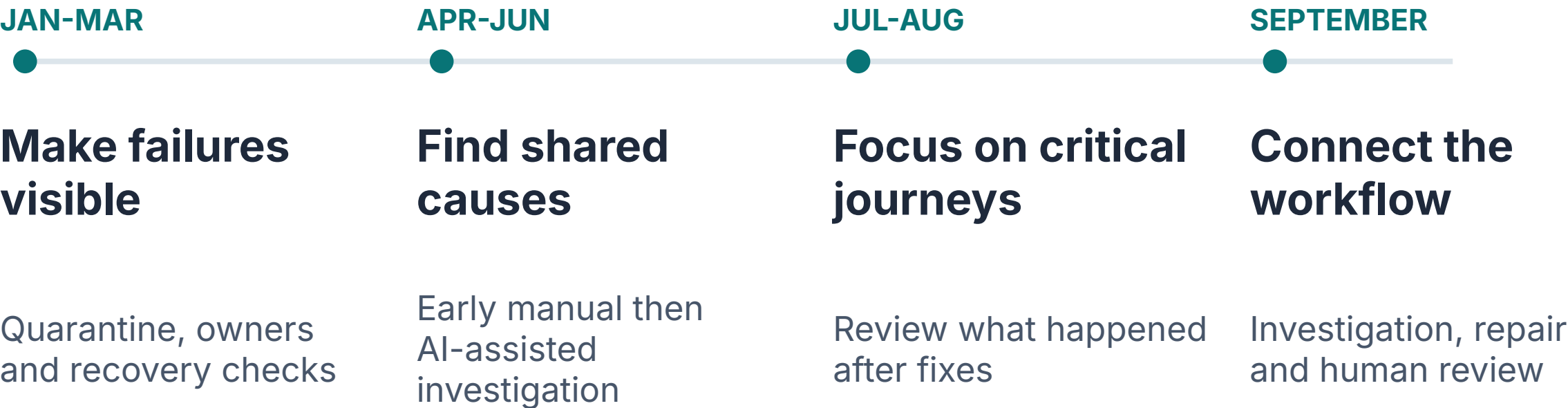
Still failing

Latest run failed;
same or different issue

Both passed their latest run.

How the changes came together

January-September 2026



These efforts overlapped.

US app: better results, with work still to do

Reported average pass rates

	January 2026 All tests	September 2026 Mission-critical + critical tests
ios	59.0%	82.6%
Android	48.5%	83.5%

Different test selections, not a like-for-like comparison.

September data through 22 September.

Source: internal January and September 2026 summaries.

Keeping results consistent took ongoing work

Run conditions changed

App changes and service outages

Repairs could be delayed

Sandbox issues and feature commitments

Core kept growing

More tests joined as journeys were mapped

QEs shape the system, not just the tests

Shape the process

Priorities and recovery checks

Question the fix

Does it still test the right thing?

Assess customer risk

With product and engineering

Start with one recurring failure

Which customer journey is affected?

Where does the failure actually begin?

What shows that the fix held?

Who will own the fix?

**Stability is not
a number you reach.**

**It is a system
you keep operating.**

Thank you.

Which failure keeps pulling your team away from feature work?

Pranav Pandit

Staff Quality Engineer | Robinhood