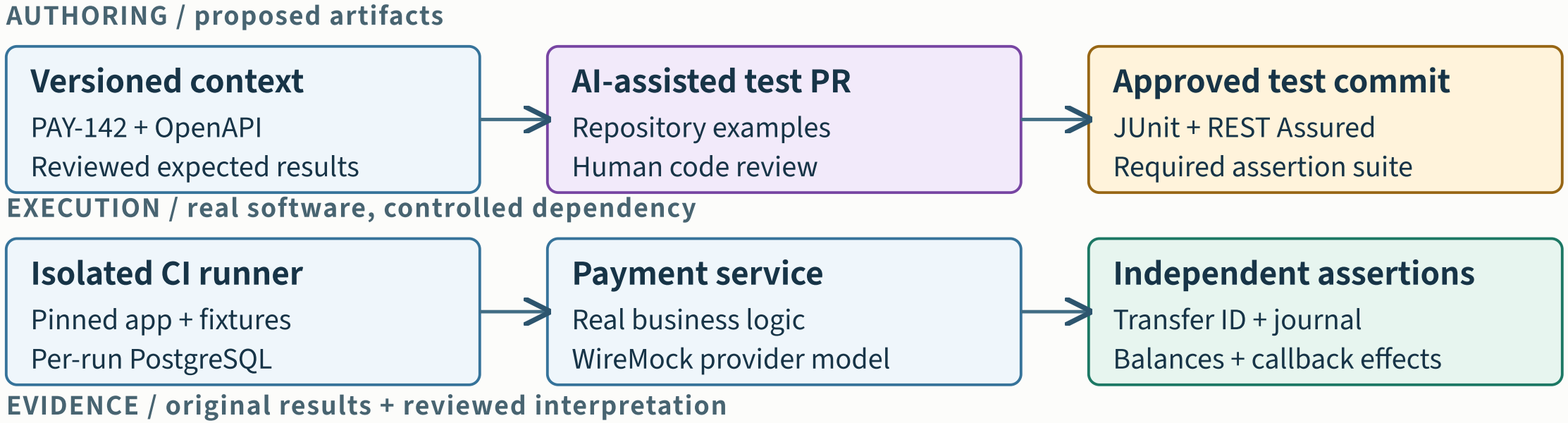
Our Banking Client: engineering blueprint

Follow one payment test from an approved contract to isolated execution and inspectable evidence.



Example stack: Azure Pipelines · Java / Spring Boot · PostgreSQL · REST Assured · WireMock.

Technical design review · Authored reference architecture; validate interfaces and tooling with the client.

Prepared by Tom Wu · AI-generated illustrations · Synthetic English narration in the web presentation.

The application landscape

Payment APIs offer the first controlled test surface. Other applications follow their own readiness.

Payment services
Java / Spring Boot
PostgreSQL journal
Versioned payment API

First pilot boundary

↔

↔

↔

Web + mobile

React journey; existing Selenium and Appium suites

Provider integration

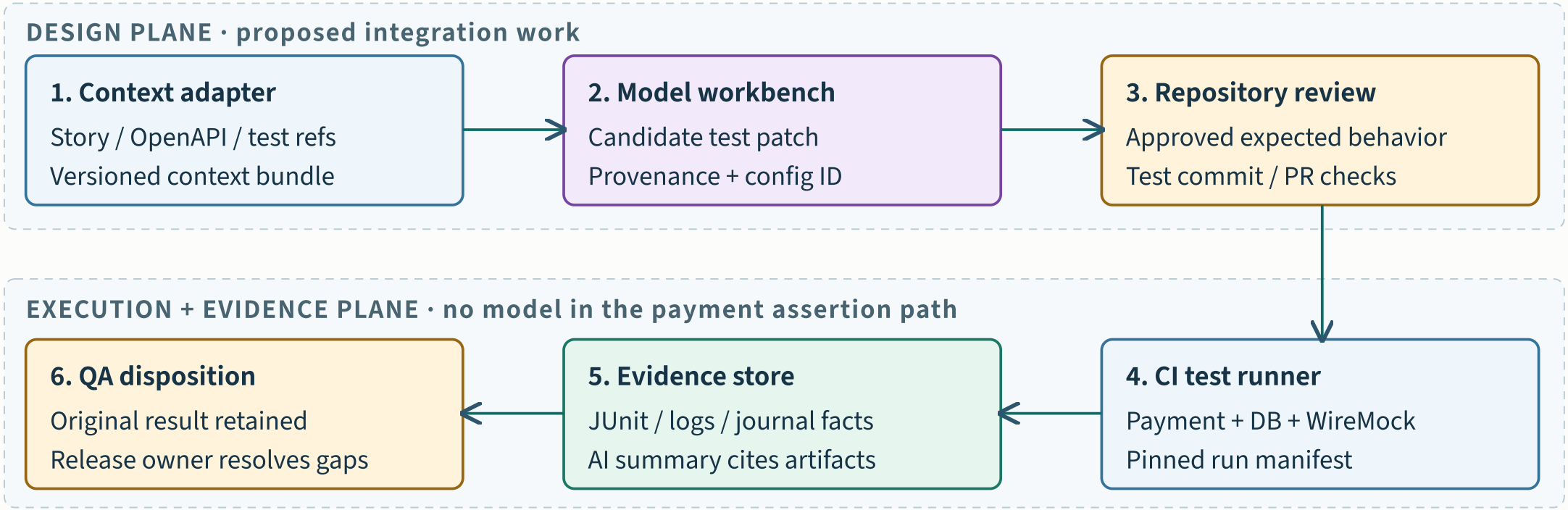
Today: limited vendor test environments; no service virtualization

Settlement batch

Legacy Java + SQL; shared nightly environment and file reconciliation

Current baseline: few shared environments and no backend virtualization. WireMock and per-run isolation are proposed pilot capabilities.

Separate authoring from test execution



Solid arrows carry versioned artifacts. AI proposes or summarizes; deterministic assertions establish the test result.

Adapters and joined evidence are integration work to build. Model output cannot change the original assertion result.

PAY-142: request contract and test oracle

Request / authored API example

```
POST /transfers
Idempotency-Key: pay-142
{
  "from": "payer", "to": "recipient",
  "currency": "CAD",
  "amountMinor": 10000
}
```

Replay + independent oracle

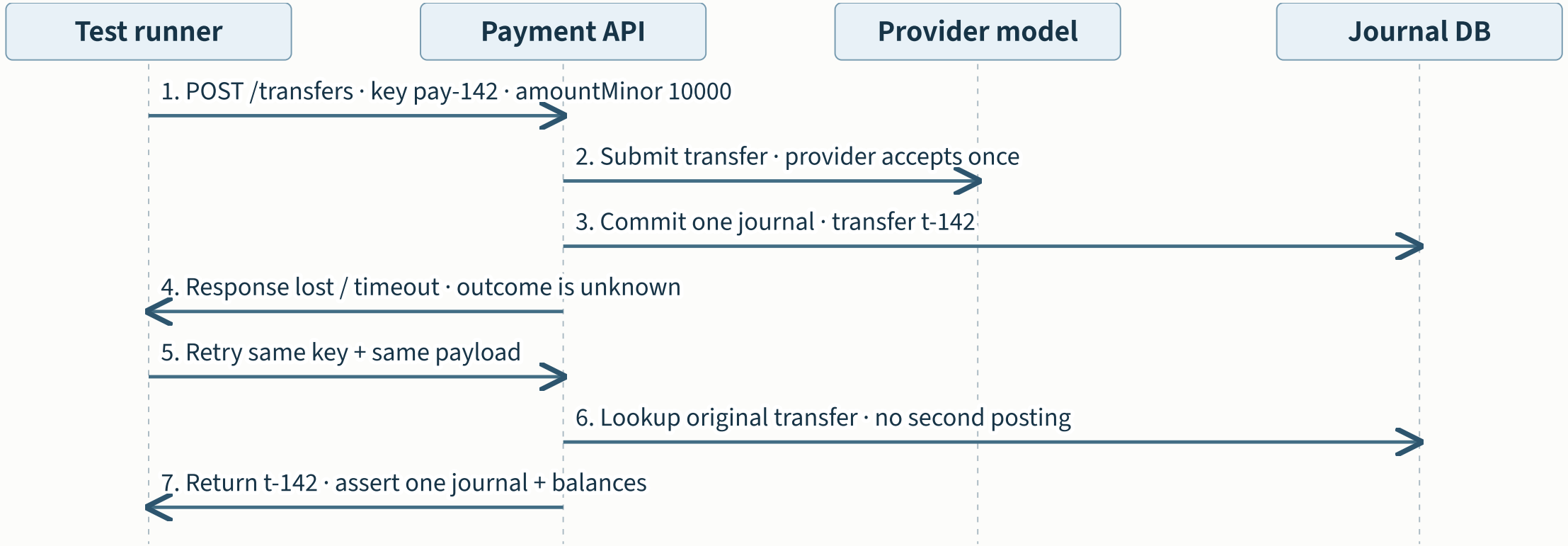
```
same key + same payload → transfer t-142
same key + changed payload → conflict

journalCount(t-142) == 1
entryCount(t-142)   == 2
payer.balance      == 90000
recipient.balance  == 10000
```

> Concurrent retries and repeated callbacks must converge on the same identity. Define key scope, retention and recovery with the service owner.

Authored contract; minor units are CAD cents. Start at 100000 / 0; no fees, FX or other account activity.

Timeout and retry: the runtime sequence



Inject the duplicate-posting variant to challenge the oracle: the journal and balance assertions must fail.

QE testing scenario. A timeout leaves the outcome unknown; a stable key must resolve to the original transfer.

Version the context before asking for a test

Input contract · authored YAML

Bind the draft to its sources

```
requirement: PAY-142@r3
contract: payments-api@v1.4
fixture: cad-transfer@v2
amount_minor: 10000
key: pay-142
open_question: key-retention-window
```

Review artifact · test matrix

Agree the expected behavior

```
same key + same payload -> original ID
same key + changed amount -> conflict
parallel retries -> one transfer
duplicate callback -> one posting
client timeout -> outcome still unknown
expired key -> rule needs domain approval
```

- AI drafts cases with source references; domain QA defines the expected ledger outcomes.
- Unresolved retention and replay rules stop approval of those cases.

Proposed artifacts for the client scenario; identifiers and conflict behavior require agreement with the API owner.

Pin and reset the test environment

Fixture contract · authored JSON

Isolate the money movement

```
{
  "run": "run-042",
  "currency": "CAD",
  "payer_minor": 100000,
  "recipient_minor": 0,
  "transfer_minor": 10000,
  "idempotency_key": "pay-142"
}
```

Run manifest · authored YAML

Pin every mutable dependency

```
service_image: sha256:<digest>
test_commit: <commit-sha>
fixture_version: cad-transfer@v2
schema_version: payments@v8
provider_stub: accept-once@v3
account_scope: isolated-per-run
preflight: health + schema + balances
fault: drop API response after commit
```

- A test fixture creates isolated accounts; shared accounts would invalidate the balance assertions.
- Testcontainers supplies disposable dependencies; WireMock simulates the provider. Real integration remains a separate check.

Proposed manifest schema. Digest and commit placeholders must resolve to real immutable versions before execution.

Assert money movement, not just HTTP status

Review anti-pattern · pseudocode

A success status is insufficient

```
retry = post_transfer("pay-142", 10000)
assert retry.status == 200

# This can pass after a duplicate debit.
# It cannot establish ledger correctness.
```

Required invariants · pseudocode

Prove identity and balances

```
assert retry.id == original_transfer_id
assert journals(retry.id).count == 1
assert entries(retry.id).count == 2
assert sum_signed_entries(retry.id) == 0
assert balance(payer) == 90000
assert balance(recipient) == 10000
```

- Capture the original accepted transfer ID from service evidence before the retry; do not derive expectations from the candidate test.
- Run the same assertions against a correct build and an injected duplicate-posting defect: pass, then fail.

Non-runnable pseudocode for this CAD 100 test scenario; a REST Assured/JUnit implementation needs approved API and ledger helpers.

The appropriate test layer

LAYER	EXAMPLE IMPLEMENTATION
Developer unit	Copilot drafts JUnit tests for idempotency logic; developers review. A fake repository cannot prove real payment integration.
Service and API	REST Assured + JUnit. Cover retry, concurrent requests, idempotency conflicts and journal outcomes.
Web journey	Playwright for the new React flow. Check pending state, retry action and the final customer message.
Existing browser and mobile	Keep owned Selenium and Appium coverage. Change frameworks only when maintenance evidence supports it.
Batch and integration	Retain settlement-file reconciliation and provider sandbox checks with explicit scheduling.

Test distribution follows behavior and observability. A new browser framework does not solve a shared settlement environment.

Publish raw CI results before AI interpretation

Azure Pipelines · task fragment

Publish failures explicitly

```
- task: PublishTestResults@2
  condition: always()
  inputs:
    testResultsFormat: JUnit
    testResultsFiles: 'results/TEST-*.xml'
    failTaskOnFailedTests: true
    failTaskOnMissingResultsFile: true
    failTaskOnFailureToPublishResults: true
```

Proposed AI adapter · authored YAML

Read evidence after the run

```
inputs: [junit, run_manifest, logs]
access: read-only
output: diagnosis.json
evidence_links: required
change_test_status: prohibited
missing_artifact: unknown
merge_or_release: human-owned
```

- Keep the runner’s failing exit status and required suites; publishing or summarizing cannot turn a failed run into success.
- Run publication even after failure. Check mandatory test coverage separately: one result file does not prove suite completeness.

Microsoft task syntax is documented; the AI adapter is proposed integration work. Job cancellation can still prevent artifact collection.

Route failures without rewriting the verdict

Diagnostic output · authored JSON

Separate observation from hypothesis

```
{
  "test": "PAY-142.retry",
  "observed": "journals=2; expected=1",
  "evidence": "run-042/ledger.json",
  "hypothesis": "replay posted twice",
  "classification": "needs-review",
  "owner": "payments-engineering"
}
```

Triage policy · pseudocode

Preserve the original test intent

```
preflight failed -> environment owner
assertion + ledger agree -> product review
unstable reproduction -> flake investigation
missing logs -> unresolved
approved fix -> new build + full rerun
original failing run -> retained
```

- Correlate test, trace and transfer IDs before grouping failures; an AI hypothesis is not a defect verdict.
- A repair proposal may change implementation or test mechanics. It may not weaken the approved money-movement assertions.

Proposed diagnostic schema and routing policy. Human review assigns the disposition; the original evidence remains available.

Define the run evidence contract

Evidence record · authored YAML

Make the decision reproducible

```
requirement: PAY-142@r3
test_commit: <commit-sha>
build_digest: sha256:<digest>
fixture: cad-transfer@v2
run: run-042
raw_results: results/TEST-payments.xml
disposition: defects/PAY-281
release_decision: pending-human-review
```

Case gate · pseudocode

Fail closed on missing proof

```
missing artifact -> UNKNOWN / block
required test skipped -> INCOMPLETE / block
ledger invariant failed -> FAILED / block
unresolved critical defect -> block
all required checks pass -> review eligible
release owner records final decision
```

- The AI summary links to retained originals; its prose is never the authoritative test result.
- Required provider integration and other release checks still apply after this API case passes.

Proposed evidence contract and pilot gate. This example does not supply a complete banking release policy.

Challenge the test with failure injection

CONTROLLED CONDITION	REQUIRED OBSERVATION
Duplicate journal defect	Second posting changes payer to 80000; journal / balance checks fail.
Same key, changed amount	Conflict returned; no additional provider submission or journal.
Concurrent same-key retries	One transfer identity and one journal after all attempts settle.
Repeated callback event	Same event ID produces no additional posting.

Author the expected behavior independently. The browser demo covers a subset; these cases require service-level implementation.

Readiness requires executable proof

CAPABILITY	EVIDENCE BEFORE AUTOMATED EXECUTION
Deploy + reset	Pinned application digest starts; a fresh run restores the same synthetic balances.
Provider control	Replay timeout and duplicate callback; identify the model revision and its owner.
Observable outcome	Read transfer identity and committed journal facts without depending on the AI summary.
Reproduction	A second engineer reruns the same manifest and obtains an explainable outcome.

Assess per application. Unknown or failed checks block the dependent execution scope; reviewed drafting has a narrower prerequisite set.

Offshore handoff and platform ownership

Each handoff carries a reproducible artifact and a named next owner.

Owner / artifact	Shared overlap	Offshore workday	Next overlap
Product + domain QA	Approve PAY-142 Rule + scenario matrix ↓	Questions wait for a named reviewer	Resolve ambiguity Updated rule → QA
Offshore QA + engineers	Accept the case Owner + review window →	Draft, review and run AI proposes; engineer approves ↓	Repair and retest Fresh run → release review
Platform + CI	Provide pinned fixtures Supported runner + reset →	Retain original evidence Run ID + failure + next owner →	Reproduce the failure Escalate missing evidence

Pilot team: one QA lead, four domain testers, two automation engineers and one data/environment engineer. Product, development and platform support contribute separately.

Correlate a failure across the runtime

Correlation record · authored JSON

Join identities across the run

```
{
  "run": "run-042",
  "test": "PAY-142.retry",
  "trace_id": "trace-042",
  "transfer_id": "t-142",
  "idempotency_key": "pay-142",
  "raw_result": "failed"
}
```

Evidence path · authored example

Trace the failed invariant

```
JUnit: journals expected 1, observed 2
  -> trace-042: two post attempts
  -> t-142: two committed journals
  -> run-042: build + fixture + stub refs

AI hypothesis: replay posted twice
QA disposition: pending owner review
```

- Correlate request attempts using trace and transfer IDs; retries remain separate events in the same scenario.
- Keep logs, journal facts and manifests as originals. Missing joins stay unresolved; redact sensitive values before AI input.

Proposed observability contract. Example identifiers are illustrative; the AI hypothesis does not override the failed assertion.

Promotion rules for a generated test

GATE	ENGINEERING ACCEPTANCE
Code + intent review	Approved behavior and independent assertions; no weakened checks or unexplained skips.
Paired challenge	Correct build passes; injected duplicate posting fails using identical fixtures.
Reproducibility	Run manifest, raw evidence and cleanup results are present; repeat failures are classified.
Suite promotion	QA owner approves the PR; required tests and separate integration checks remain enforced.

Do not promote a generated test merely because it compiles or passes once. Benchmark pilot effort separately.

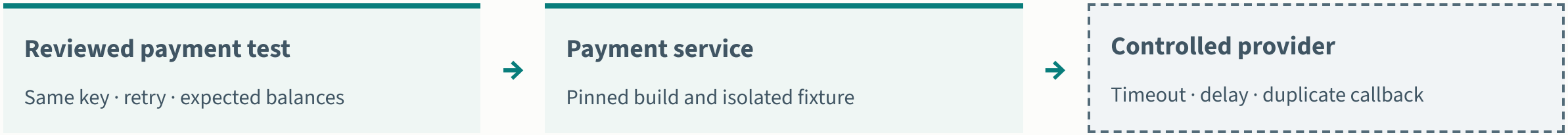
Map foundation gaps to blocked capabilities

REQUIRED FOUNDATION	CAPABILITY UNLOCKED / CONSEQUENCE IF MISSING
Approved rules + repository context	Reviewed test drafting; without a trustworthy oracle, keep cases as unapproved proposals.
Isolated runner + resettable fixtures	Automated service execution; shared mutable data prevents repeatable comparisons.
Provider models + contract owner	Deterministic negative-path tests; unknown model fidelity needs real integration evidence.
Run IDs + raw results + named owner	Evidence-linked diagnosis and team reuse; missing evidence routes to investigation.

Foundation work starts with the pilot. Existing DevOps and QE maturity determine which capability can progress.

Service virtualization and real integration

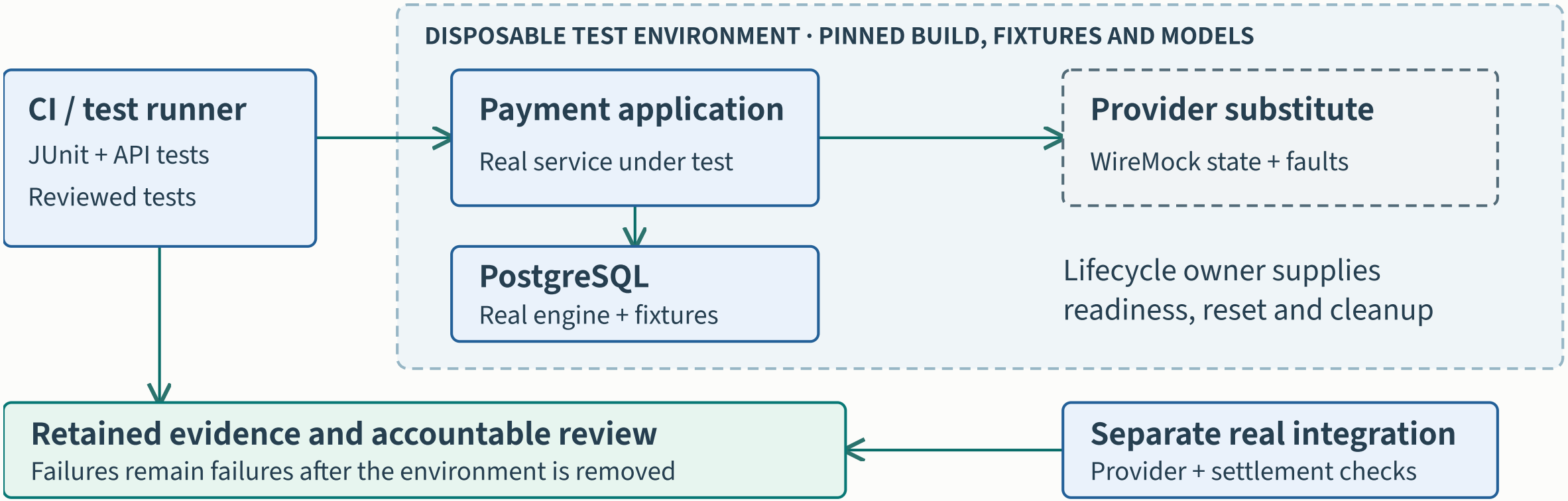
Control the payment provider for repeatable tests; keep a separate real-integration gate.



Proposed payment test architecture. A passing substitute is evidence about the modeled behavior, not proof of compatibility with every real dependency. Dependency strategy and sources →

WireMock is an HTTP example. Contract fidelity, callback behavior and batch access need explicit owners; passing a stub does not prove end-to-end correctness.

Our Banking Client: test environment



The CI runner checks the real payment service and database against reviewed provider models. Retain separate provider and settlement evidence.

Technology roles in the payment pilot

RUN

Containerization

Package and run real software with repeatable dependencies.

EXAMPLE

A PostgreSQL instance created for one test run.

SIMULATE

Service virtualization

Replace selected dependency behavior with a controlled model.

EXAMPLE

A WireMock provider with an intentional timeout.

CHECK

Contract testing

Check whether consumer and provider interactions agree.

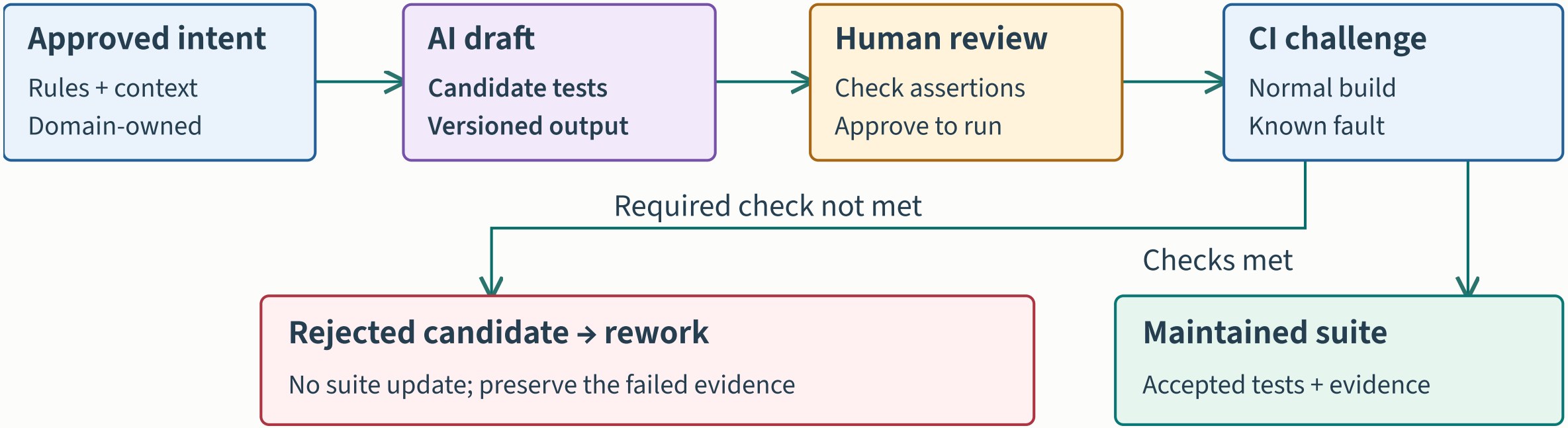
EXAMPLE

A provider change fails a reviewed Pact contract.

These capabilities complement each other. They do not establish complete business correctness or replace all real integration checks. Technology choices and limitations

Choose Testcontainers, Compose or an existing supported environment for the test scope. Mobile devices and legacy settlement retain their own integration requirements.

A candidate test must earn acceptance



PAY-142 must catch an incorrect transfer, journal or balance. The guided example rejects the candidate and leaves the maintained suite unchanged.

Instrument the pilot at run level

Run event · authored JSON

Capture comparable observations

```
{
  "run": "run-042",
  "mode": "assisted",
  "scope": "PAY-142-api",
  "raw_result": "failed",
  "review_minutes": "not-recorded",
  "model_cost": "not-recorded",
  "rerun_count": "not-recorded"
}
```

Pilot measures · definitions

Keep quality and effort visible

```
defect challenge: detected / injected
case completion: executed / required
flake rate: unstable / repeated tests
review load: human minutes per case
time to disposition: failure to owner decision
net effort: draft + review + rerun + operation
```

- Compare baseline and assisted runs with the same scope, environment and acceptance rules; separate setup from recurring effort.
- Set pilot thresholds after a baseline exists. Do not infer headcount savings from generated-test volume or model speed.

Authored measurement schema; no pilot timing, cost or effectiveness observations have been recorded.

The runtime from setup to cleanup

Proposed execution lifecycle. Numbered steps show order, not elapsed time.

1. **Create and check**
   Pin the build, tests and configuration. Provision supported resources and wait for health checks.

2. **Seed and configure**
   Create isolated synthetic accounts. Load the owned, versioned virtual-provider behavior.

3. **Execute and challenge**
   Run required tests. The deliberate duplicate-posting variant must fail its independent assertion.

4. **Retain and review**
   Keep original assertions, setup diagnostics and run IDs outside disposable resources. Review omissions.

5. **Clean up on every path**
   Attempt teardown after success or failure. Escalate failed cleanup to the named platform owner.

6. **Reconcile integration**
   Check the real provider and settlement separately. Contract drift triggers model review and rerun.

> Evidence survives cleanup. A missing or failed required check holds the release.

Repeatable execution is a platform capability. Retain failure evidence and attempt cleanup on every path; real integration remains separate.

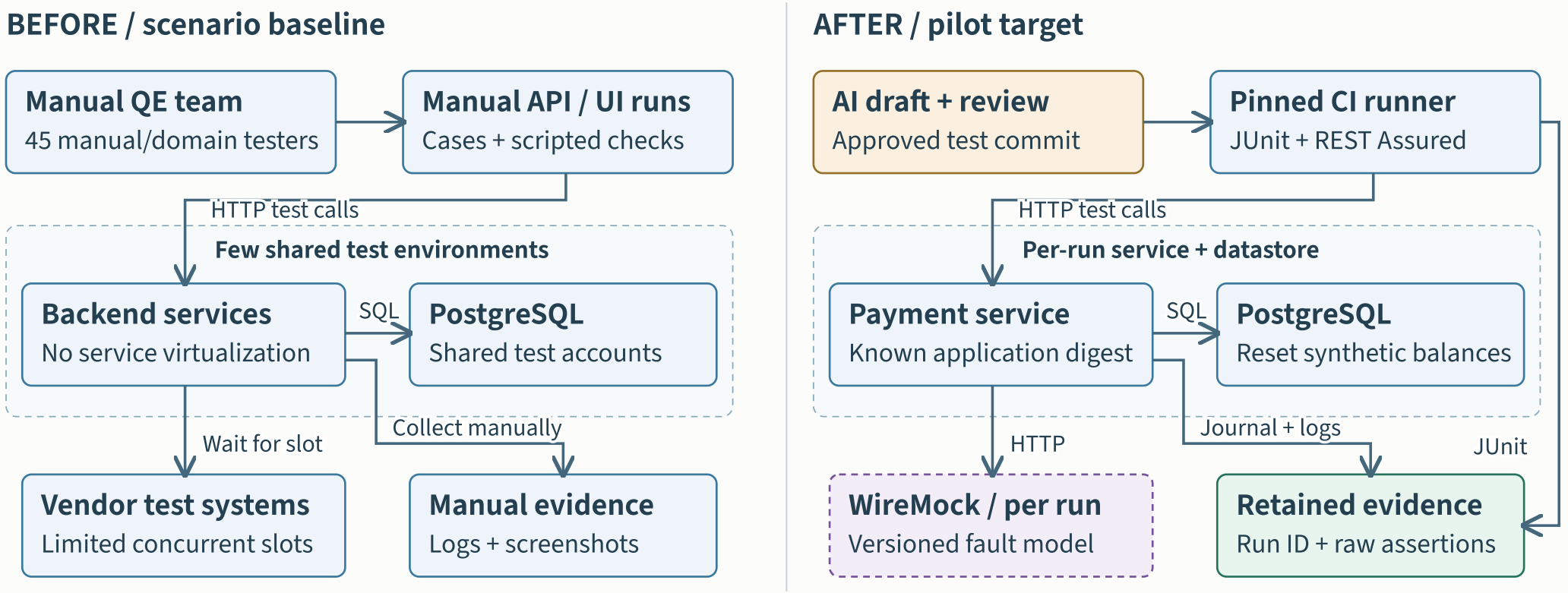
Integration adapters have explicit contracts

Reuse CI execution and artifact storage. Add owned context, model and evidence adapters around those systems.

ADAPTER / OWNER	INPUT → OUTPUT / FAILURE HANDLING
Context / QE platform	Story + API + test revisions → bundle; unresolved rules block approval of affected cases.
Model / AI platform	Bundle + configuration → patch; timeout or quota → no candidate, bounded retry.
Runner / application QA	Approved test commit + manifest → raw results; preserve exit status, retain evidence, clean up.
Evidence / QE platform	Run artifacts → indexed links; incomplete joins stay unknown; AI report is read-only.

Proposed adapter contracts. Use request IDs for safe retries and record attempts; platform owners define timeouts and retention.

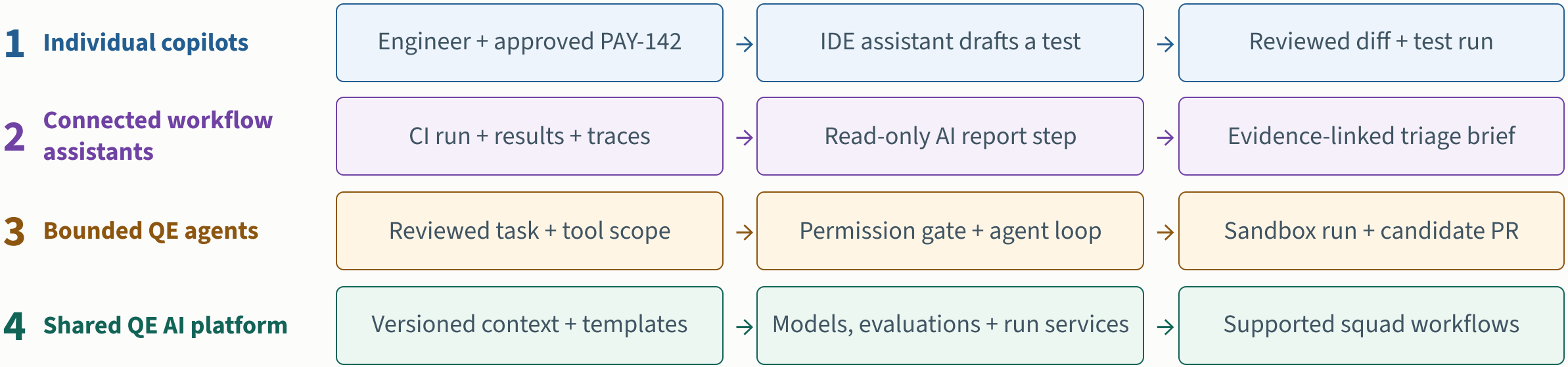
Before / after: the API test architecture



Target: virtualized service tests reduce dependence on vendor slots. Real-provider and settlement checks remain separate gates.

75 offshore QA staff share constrained environments. Proposed isolation and virtualized dependencies support parallel service tests; real integration stays scheduled.

How the four AI adoption layers connect



Parallel integration paths, not four consecutive runtime steps. Layer 4 supplies shared services to layers 1–3.

Proposed integration paths. The CI assistant is read-only; bounded agents need isolated tools, independent checks and human approval.

Measure test effectiveness and decision evidence

MEASURE / PROPOSED ACCEPTANCE	CALCULATION / EXPLICIT DENOMINATOR
Catch payment faults Target: Catch every critical seeded fault	Correctly detected viable critical payment faults / all approved viable critical faults in the seeded catalogue. Unexecuted faults remain in the denominator; report separately.
Cover critical journeys Target: Exercise every critical scenario	Approved critical scenarios with reviewed assertions and a valid execution result / all approved critical scenarios in the frozen pilot matrix. Skipped or blocked scenarios are incomplete.
Explain each release decision Target: Complete every decision record	Pilot release-candidate decision packs containing every required, resolvable evidence field / all pilot release candidates submitted for a decision. Missing packs remain in the denominator.

> Baseline and observed after: Not recorded. Freeze scope and definitions before comparing.

A clean control must pass; the intended injected-fault assertion must fail. Coverage and complete evidence do not imply a release pass.

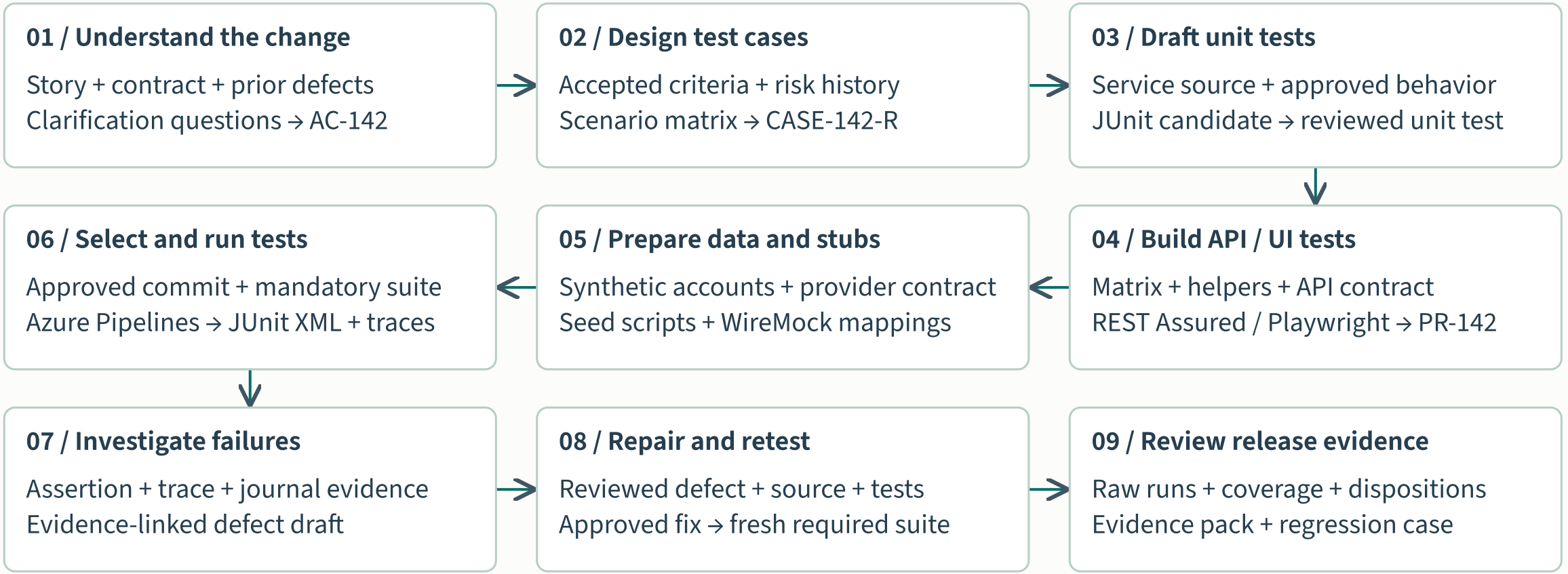
Prove reliability, integration and reuse

MEASURE / PROPOSED ACCEPTANCE	CALCULATION / EXPLICIT DENOMINATOR
Trust repeat runs Target: No unexplained critical flakes	Test cases with mixed pass/fail results / all cases completing the agreed unchanged-configuration repeat protocol. Publish incomplete cases / all selected cases separately.
Validate real integrations Target: Verify agreed provider behaviors	Approved provider behaviors verified against the real provider sandbox / all provider behaviors assigned to the agreed integration check. Report stub outcomes separately.
Enable the next squad Target: Second squad runs independently	Journeys independently configured, run and evidenced by the second squad / all journeys assigned to its agreed reuse trial. Record pilot-author interventions separately.

> Baseline and observed after: Not recorded. Freeze scope and definitions before comparing.

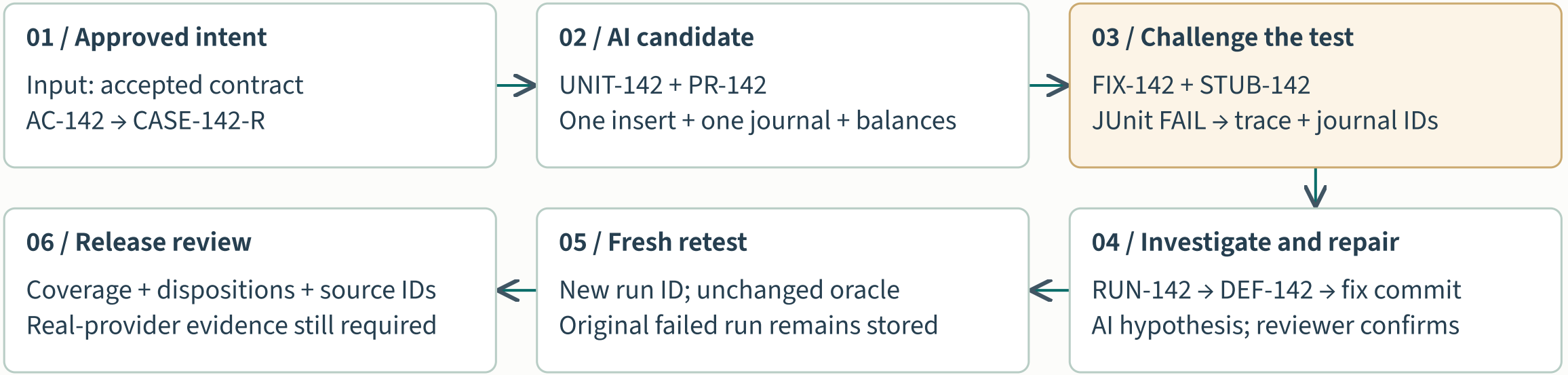
A passing retry never erases failure. Stubs do not prove real integration. Agree the independence of the reuse trial before measuring.

Nine AI-assisted steps, with explicit handoffs



AI drafts; reviewers approve; CI executes. Context, artifacts and evidence remain versioned.

PAY-142: requirement, test, failure, fix and proof



Authored trace IDs. Fixed assertions: original transfer ID, one journal, payer 90000 and recipient 10000 minor units.

Three states: isolate the incremental AI contribution

01 / Manual QE

Manual API requests / spreadsheets
Shared environments; no provider model
QA collects and reconciles evidence

→

02 / Modernized QE

REST Assured / JUnit + CI
Testcontainers + validated WireMock
Raw assertions, traces and run IDs

→

03 / AI-assisted QE

Approved context → AI candidates
Reviewed PRs → existing CI runners
Evidence → AI summary → owner

Establish the baseline

Active work · vendor waits · coverage gaps

Prove repeatability

Environment readiness · reruns · raw evidence

Isolate added AI value

Candidate acceptance · review · rework

Compare equivalent payment packs. Retain real-provider checks and measure review, rework and platform overhead.

Brainstorm questions: API architecture

1. Where is idempotency enforced: API, provider request, journal transaction and callback handler?
2. Which artifact proves one posting when the response is lost or the callback is duplicated?
3. Can two engineers reproduce the same failure from a pinned manifest and fresh fixtures?
4. Who owns provider-model drift, failed cleanup and evidence retention?

Resolve the contract, repeatability, ownership and evidence questions with system owners before expanding execution authority.