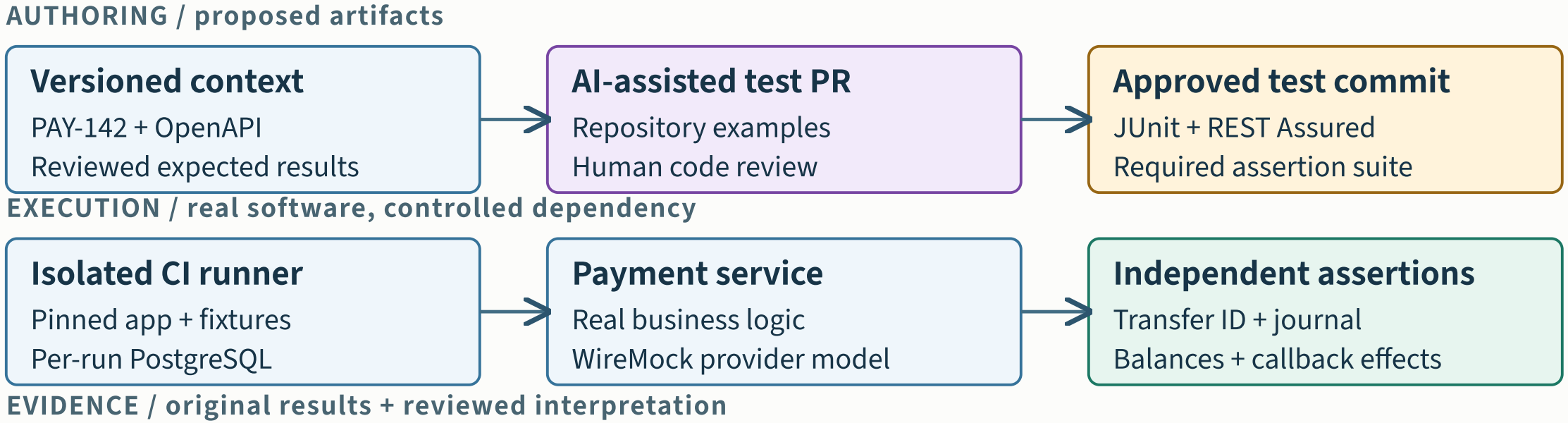
Our Banking Client: engineering blueprint

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

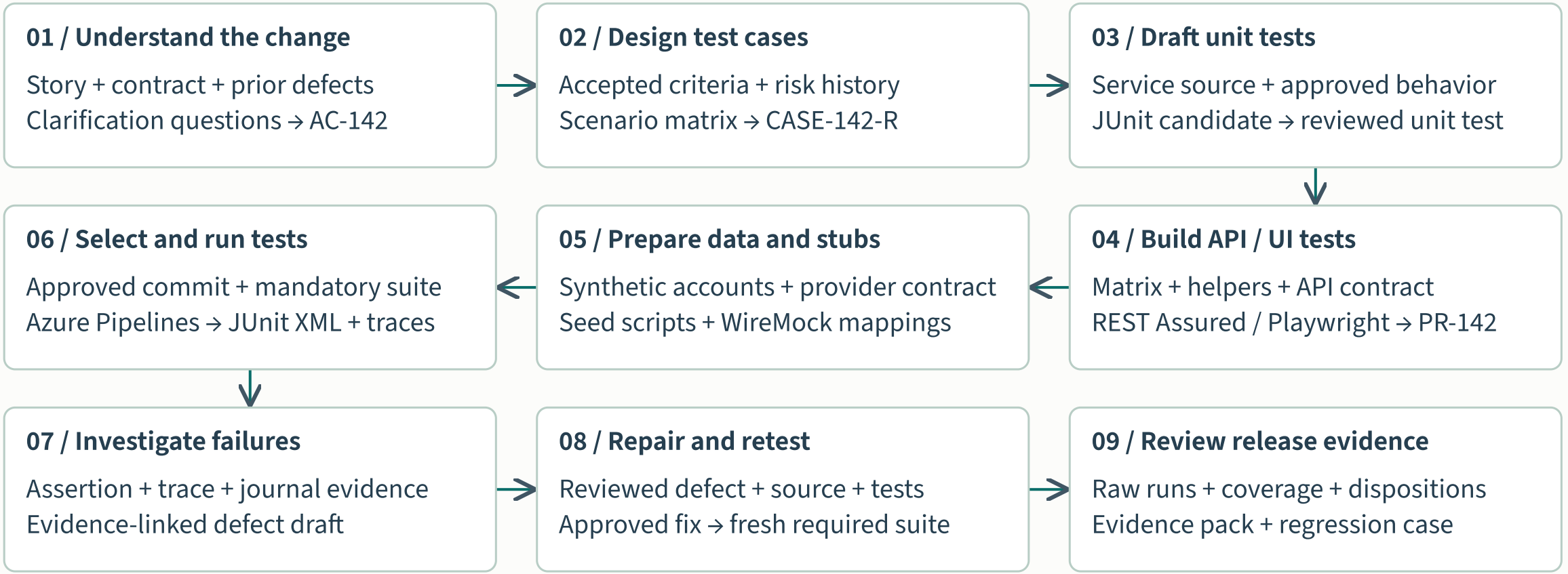


Example stack: Azure Pipelines · Java / Spring Boot · PostgreSQL · REST Assured · WireMock.

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.

Nine AI-assisted steps, with explicit handoffs



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

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.

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.

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.

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.

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.

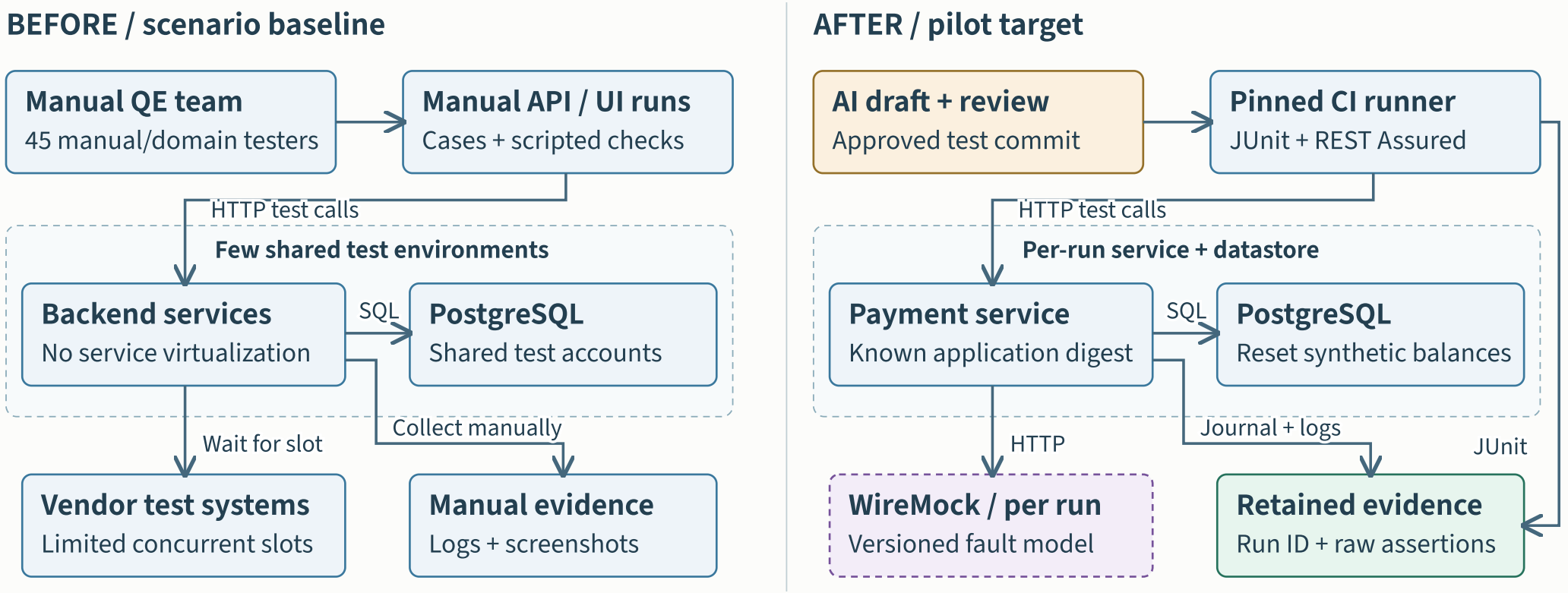
Integration adapters have explicit contracts

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.

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.

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.

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.