LLM-Assisted BDD Functional Testing with Human-in-the-Loop Control

From Business Requirements to Executable Tests —
Safely and Deterministically

RITIKA GUPTA
SAMPLE WEB APPLICATION

PROBLEM STATEMENT

Problem to Solve - Current State

- Business requirements are typically written in plain English, making them ambiguous.
- Test scenarios are manually written, leading to inconsistent and error-prone coverage.
- Blind LLM automation is inherently unsafe for direct test execution due to hallucinations.

Goal - Target State

- Design a system that converts requirements directly into structured BDD Gherkin.
- Automate only safe happy paths that pass strict validation checks.
- Maintain a mandatory human approval gate before any execution occurs.

Why LLMs Alone Are Not Enough

Hallucinated Steps

LLMs may invent UI elements (buttons, fields) that do not exist in the application, leading to flaky failures.

Unsupported Actions

Generating complex interactions (e.g., "drag-and-drop") that the underlying framework cannot execute reliably.

Direct Execution

Executing raw LLM output without validation risks triggering destructive actions in staging environments.

No Audit Trail

Black-box generation leaves no trace of why a test was created, making debugging and compliance impossible.

System Architecture

Generation ≠ Execution

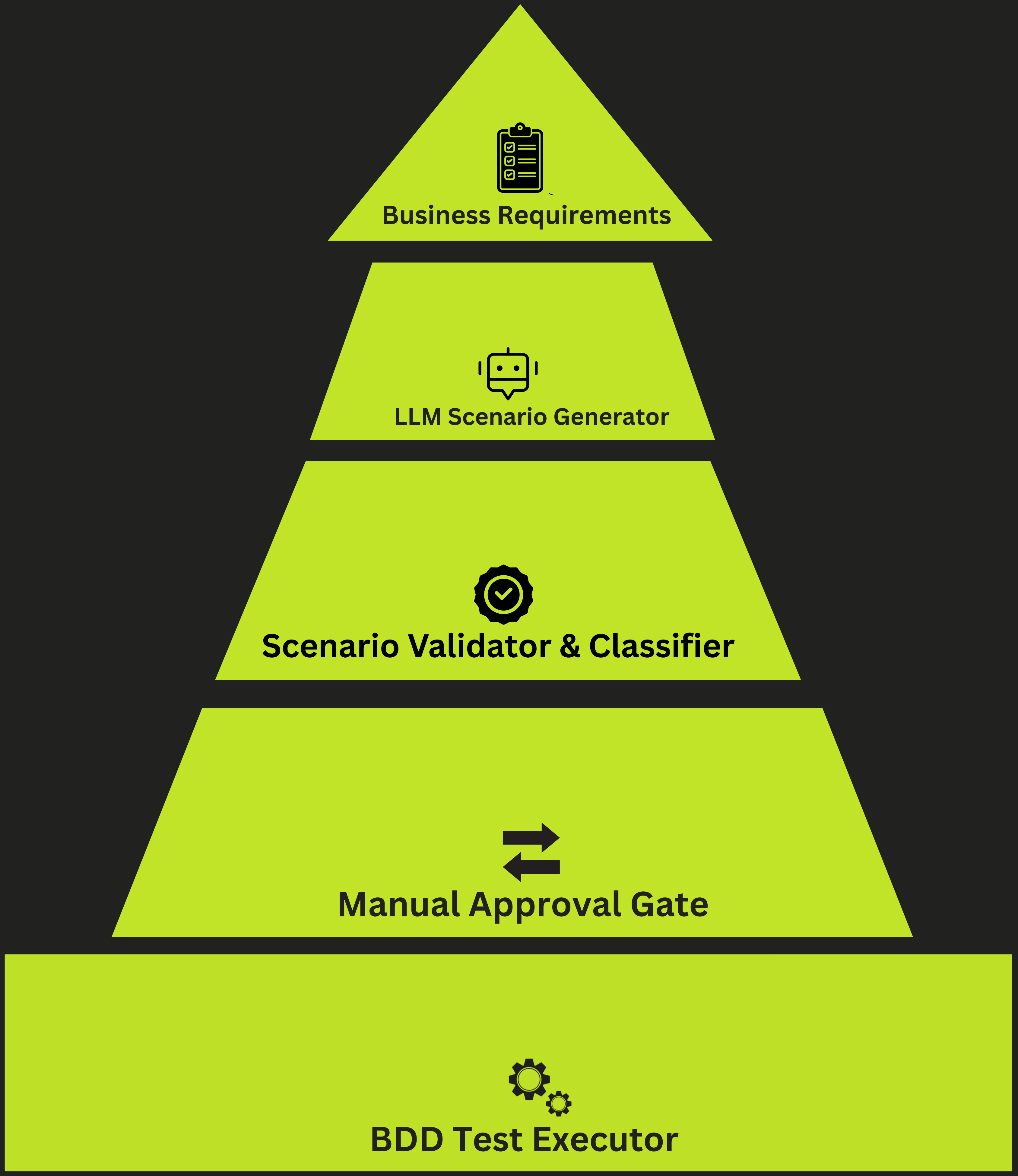
The AI creates the blueprint, but never touches the execution engine directly. This decoupling prevents hallucinated actions from running on live systems.

Deterministic Validation

Rules-based syntax checking ensures only valid Gherkin proceeds.

Human-in-the-Loop Control

Explicit approval required. No silent failures or silent executions.



Business Requirements

LLM Scenario Generator

Scenario Validator & Classifier

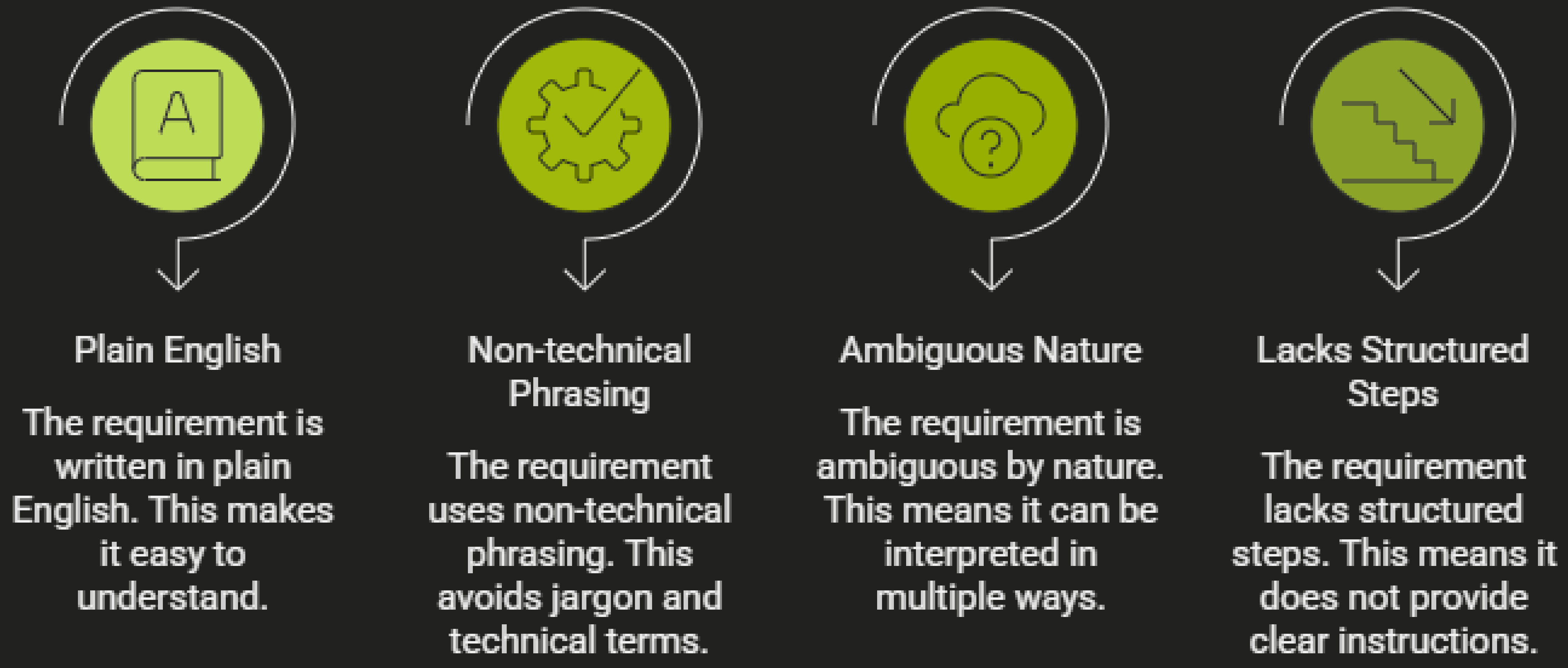
Manual Approval Gate

BDD Test Executor

Business Requirement Input

Processing raw human language into structured test logic

Input Characteristics



Plain English

The requirement is written in plain English. This makes it easy to understand.

Non-technical Phrasing

The requirement uses non-technical phrasing. This avoids jargon and technical terms.

Ambiguous Nature

The requirement is ambiguous by nature. This means it can be interpreted in multiple ways.

Lacks Structured Steps

The requirement lacks structured steps. This means it does not provide clear instructions.

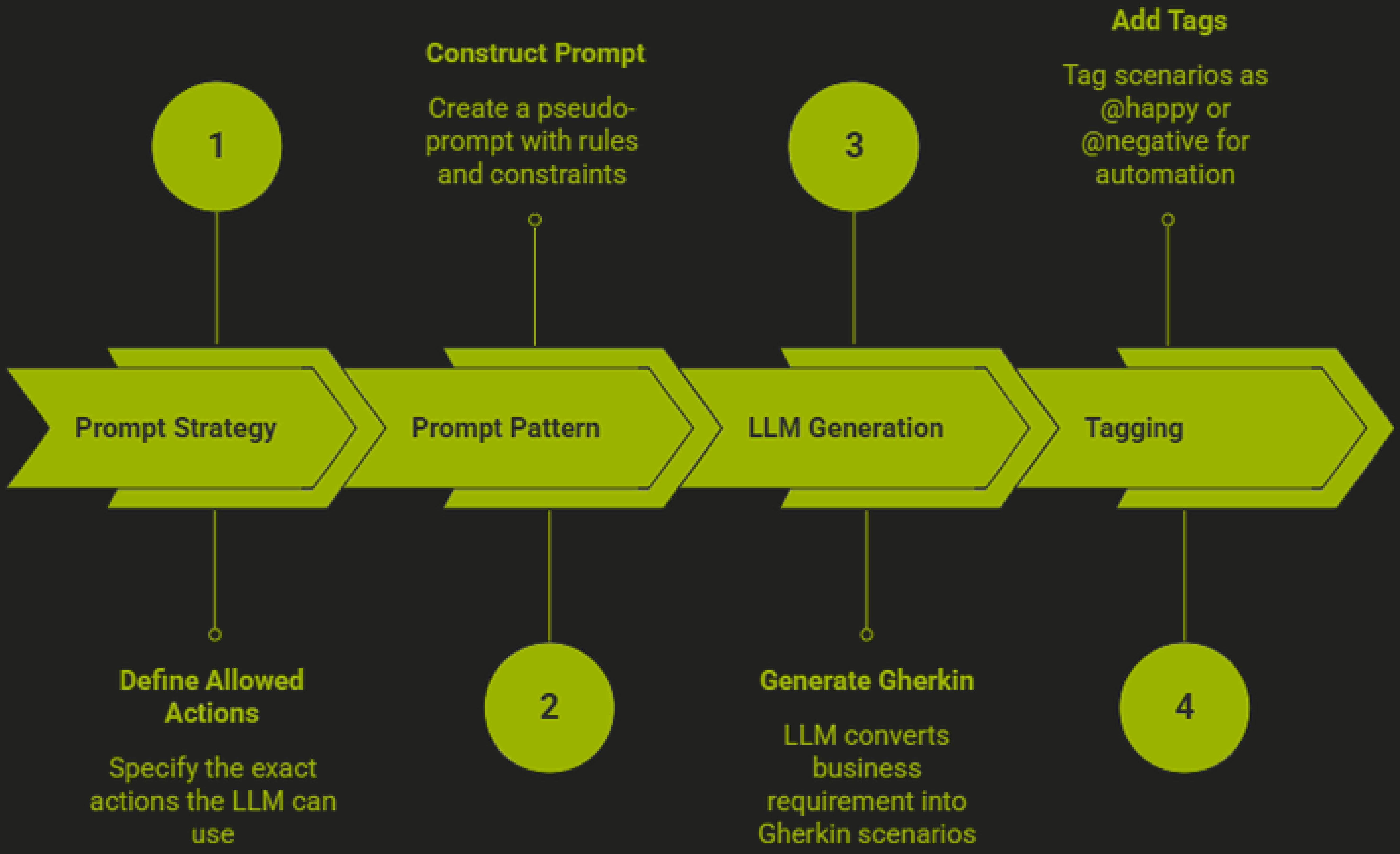
Design Choice

- No assumptions hardcoded
- Interpretations delegated to scenario generation
- Respects original ambiguity

Raw Requirement Example

“Users should be able to log in using valid credentials. Invalid credentials should show an error.”

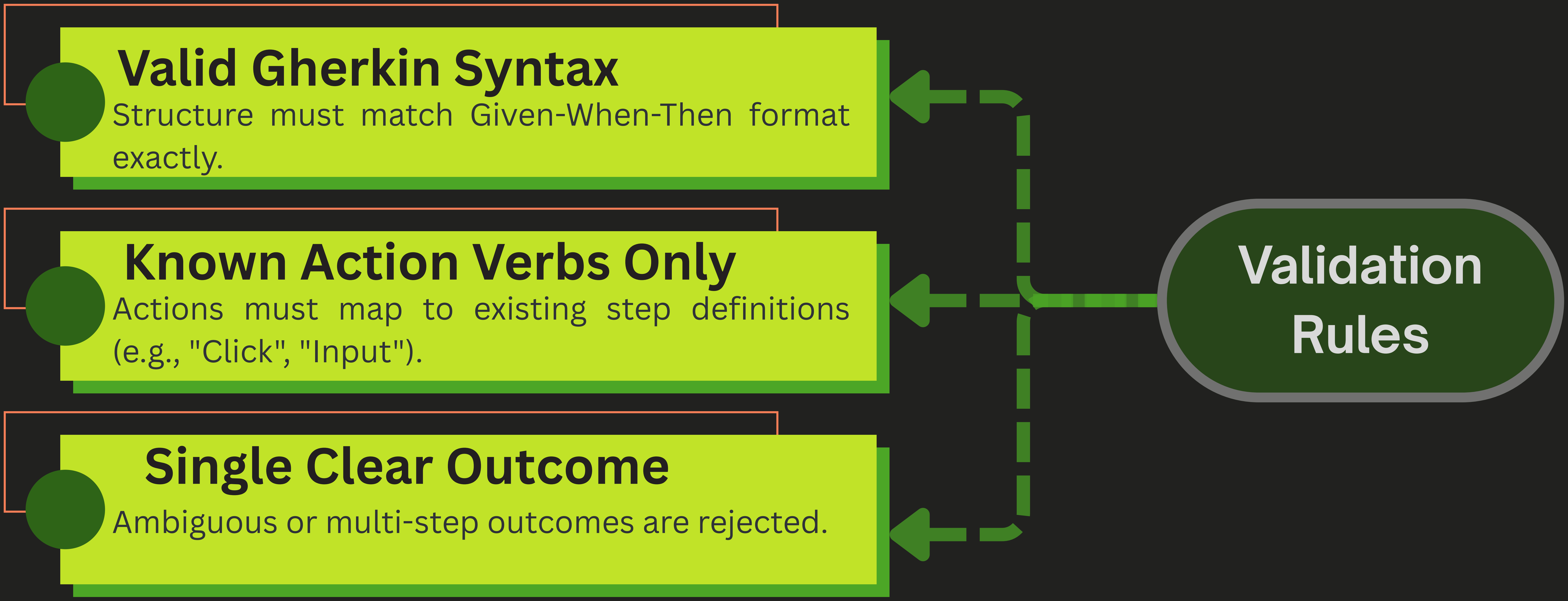
LLM-Based Gherkin Generation (Controlled)



1
Prompt Strategy
Define Allowed Actions
Specify the exact actions the LLM can use
Construct Prompt
Create a pseudo-prompt with rules and constraints
Prompt Pattern
2
3
LLM Generation
Generate Gherkin
LLM converts business requirement into Gherkin scenarios
Add Tags
Tag scenarios as @happy or @negative for automation
Tagging
4

Scenario Validation (Critical Safety Layer)

Ensuring generated tests are safe before they ever reach execution.



Valid Gherkin Syntax

Structure must match Given-When-Then format exactly.

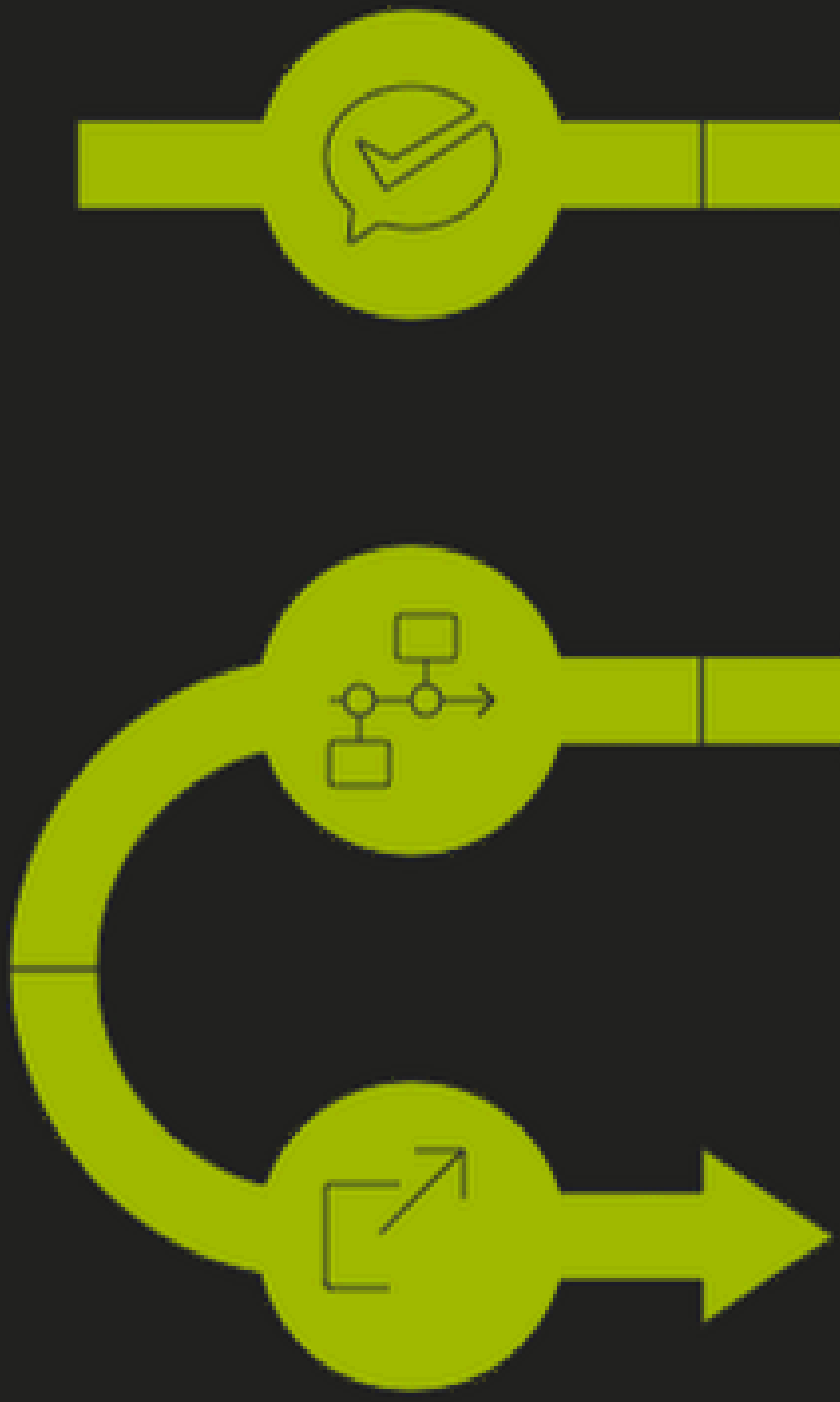
Known Action Verbs Only

Actions must map to existing step definitions (e.g., "Click", "Input").

Single Clear Outcome

Ambiguous or multi-step outcomes are rejected.

Happy Path Selection Logic



Scenarios with "@happy" tag

Identify happy scenarios

Scenarios with "PASS" validation status

Filter for successful scenarios

Happy path selection

Automatic selection of happy paths

Negative paths generated

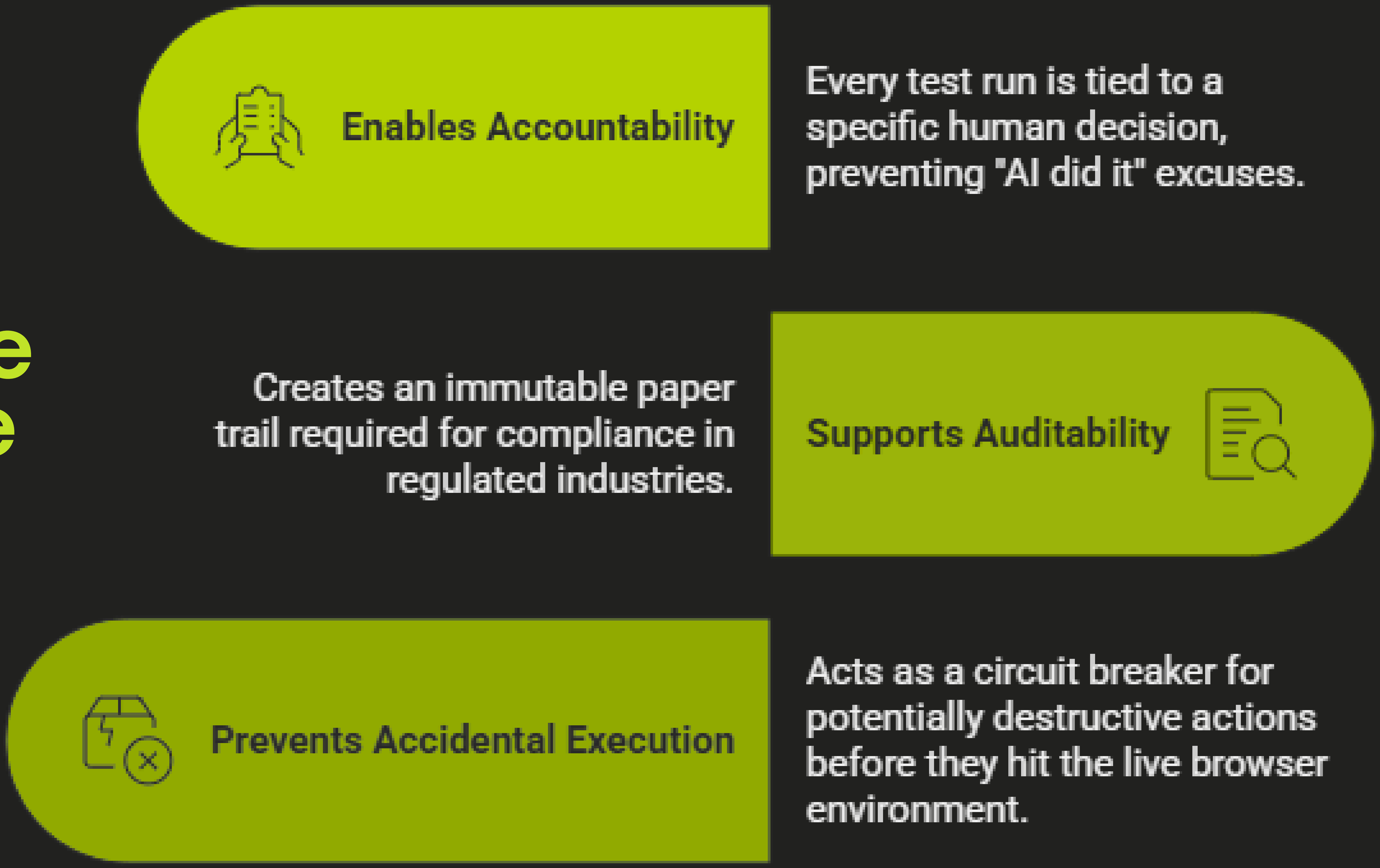
Create negative paths

Negative paths not executed

Ensure negative paths are not executed

Manual Approval Gate (Human-in-the-Loop)

Workflow Lock-In



Enables Accountability

Every test run is tied to a specific human decision, preventing "AI did it" excuses.

Supports Auditability

Creates an immutable paper trail required for compliance in regulated industries.

Prevents Accidental Execution

Acts as a circuit breaker for potentially destructive actions before they hit the live browser environment.

End-to-End Output Summary

Verifiable artifacts produced at every stage of the pipeline

Generated Gherkin Scenarios

Generated generated.feature files (Happy/Negative paths)

Validation Report

Syntax checks & action capability verification

Manual Approval Record

Audit log with timestamp & approver ID

Automated Execution Results

Pass/Fail status logs & screenshots.

THANK YOU.