ORACLE

# Testing Agentic Applications

## Why AI needs a new approach to software testing

October 2026, Version 1.0
Copyright © 2026, Oracle and/or its affiliates
Public

## Executive summary

As Oracle Fusion Agentic Applications move into production, the teams building them face a common challenge: The testing approach that made deterministic software safe to ship does not fit software that reasons. An application that interprets a request, retrieves enterprise information, invokes tools, routes through a workflow, and generates an answer can return different responses on two consecutive runs and still be entirely correct. Comparing these responses to an expected string may identify a failure when the application is working as designed.

This paper describes how the Agentic Testing and Lifecycle Automation Suite (ATLAS) in Oracle AI Agent Studio supports testing of Fusion Agentic Applications while preserving the intelligence that needs to be tested. Teams can use evaluated scenarios and supporting evidence to help assess application behavior, reduce dependence on changing enterprise data during regression testing, and compare models or configurations based on evidence. Testing becomes a mechanism for helping to support and validate agentic development.

This paper focuses on workflow testing and lifecycle automation in AI Agent Studio. Replay-based regression testing demonstrates behavior against known enterprise context and complements live integration validation of the current environment. Testing of app panels and the Ask Oracle plugin are outside the scope of this paper.

> ✓ **The core challenge**
>
> **How do we test an application whose behavior is partly deterministic and partly intelligent?**

## Why the traditional testing contract breaks

Traditional software testing rests on a simple contract: Provide an input, run the application, and compare the result to an expected output. That works when a function calculating an invoice total or applying a business rule should produce the same result whenever its inputs and system state are the same. Agentic applications break this contract because their behavior depends on more than code. It also depends on the model, prompt, enterprise data, retrieved context, tool responses, workflow route, and reasoning performed along the way. Two runs can produce different responses, and both can be correct.

Consider a manager who asks, “Review my organization’s performance information and tell me where I should focus my attention.” Answering may require the application to identify the manager’s organization, retrieve employee and performance records, iterate through them, apply business rules, invoke a model, and synthesize a recommendation. One run might report that “three employees require additional management attention,” while another says “three team members may benefit from closer follow-up.” A string comparison identifies those as different responses. A human reviewer considers them equivalent.

The data behind the answer adds a second source of variation. For example, the manager had 10 reports when the test was written and 11 when it runs a week later. Because a performance record changed, a row that exists in development is absent from the test environment. The response changes even though the workflow is functioning correctly. When models, live enterprise data, retrieval, tools, and orchestration interact, a failed end-to-end test can be hard to interpret: Did the application regress, did the data change, did the workflow take a different route, or did the model simply phrase the same answer differently? ATLAS helps teams answer that question.


Copyright © 2026, Oracle and/or its affiliates / Public

ORACLE

# Finding the boundary between repeatable and real

One way to make an agentic test repeatable is to mock everything: data, services, retrieval, child workflows, models, agents. The test becomes stable, but that stability limits what the test can demonstrate. If the model never reasons, its reasoning is untested. If workflow decisions are predetermined, orchestration is untested. Taken far enough, the test validates the mocks rather than the application.

The opposite extreme keeps all data, services, retrieval, and models live. That is realistic, but too many variables change at once. A test that answered one way yesterday and another today might be reacting to a code change or to data that changed overnight. The more variables change together, the less a result tells you.

ATLAS draws the smallest practical boundary needed to stabilize a test. Supported data-producing operations at selected boundaries, such as enterprise responses, service calls, and retrieval results, are captured once and replayed on later runs. Outside those replay boundaries, workflow execution continues. Conditions evaluate, switches route, transformations run, prompts are assembled, and the model and agent nodes produce fresh output. The surrounding context is held constant while the intelligent behavior remains observable.

Consider the manager application again. On an initial run, the response containing those 10 employees is captured. On later runs, ATLAS replays it instead of relying on the data available that day. The workflow processes the same 10 employees, rebuilds the prompt, runs its logic, invokes the model, and produces an answer for evaluation. The enterprise context is fixed; the behavior is live.

Replay boundaries define the coverage of each test. A parent workflow can replay the result of a child-workflow, and a test at a LOOP boundary can replay a complete output without executing the LOOP body. Separate tests of child workflows and internal operations cover the work behind those boundaries. File replay does not revalidate the live integration, and a missing required fixture for external data stops replay before the data call.

**✓ The principle**

**Replay the context but execute the intelligence.**

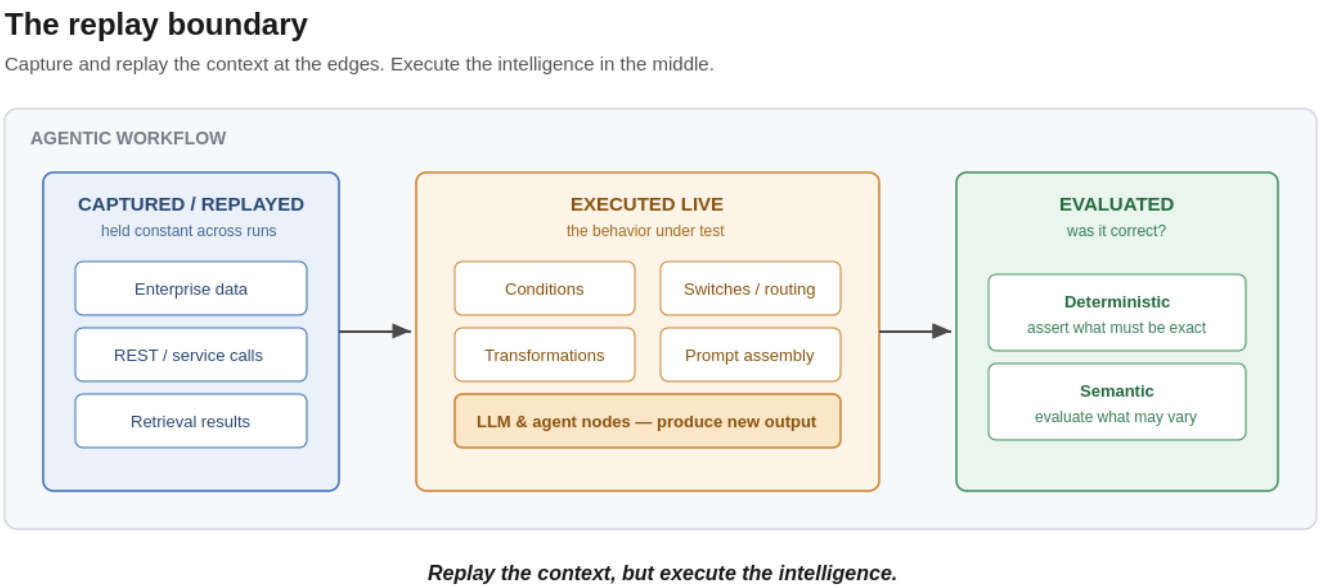


**Figure 1.** ATLAS captures and replays context at workflow boundaries while the reasoning executes live, then evaluates the result deterministically and semantically.


Copyright © 2026, Oracle and/or its affiliates / Public

## From test cases to meaningful scenarios

Scenarios change how a test case is defined. In deterministic software, a test often consists of an input and an expected output. For an Agentic Application, the scenario is the unit of testing. A useful test describes the situation being created, the input that initiates it, the enterprise context available, the expected workflow behavior, and the criteria used to judge the result.

Consider a workflow with a normal manager path plus branches for a missing manager, missing evaluation data, missing ratings, and an escalation condition. These are five business situations the application must handle correctly, not five expected responses. ATLAS analyzes a workflow to surface meaningful scenarios, such as primary execution paths, application stages, trigger variations, condition and switch outcomes, fallback branches, and data-dependent variations. Developers can add their own scenarios to preserve specific business cases or regression coverage.

> ✓ **The shift**
>
> **Stop asking "How many tests do we have?" Instead, ask "Which distinct behaviors have we demonstrated?"**

The inputs matter as much as the paths. A REST-triggered workflow needs parameters; an Agentic Application query stage needs a realistic user question; a specific switch branch needs a carefully chosen value. ATLAS uses workflow and trigger metadata, run profiles, routing requirements, developer instructions, and model-assisted generation to construct inputs appropriate to each scenario. Execution provides evidence of whether the intended path was reached. A smaller set of well-constructed scenarios gives more confidence than many superficially different prompts that all exercise the same route.

## Natural language starts the testing flow

Developers work through an AI coding assistant such as Codex or Claude. A request can describe the desired coverage and completion criteria. For example, "Create regression tests for the team performance workflow. Cover the manager happy path, missing manager, missing evaluation, and missing ratings. Use stable replay data where possible, run the tests, and show the results."

The assistant selects the required ATLAS operations to discover scenarios, create or update tests, prepare data, execute checks, complete evaluation, and report outcomes. A preview request shows the plan without creating or running tests. Existing tests can also be run without regeneration.

Inputs must fit the scenario. A REST trigger needs explicit parameters; a query stage needs a realistic question; a switch branch needs an input that reaches its outcome. Workflow metadata, run profiles, developer instructions, and observed execution provide the evidence for choosing and validating those inputs.

## Making enterprise test data portable

When the scenario is the unit of testing, enterprise data becomes part of it. Test environments rarely contain identical business data. A test built around a specific manager and employee may pass in one environment and fail in another simply because the record does not exist in both environments. The same problem occurs across every functional area. A procurement test tied to a particular supplier, requisition, and invoice, or a service test tied to a specific request, is only as portable as the data it assumes.

ATLAS lets the data a scenario needs travel with the test, subject to applicable data protection requirements. Responses captured at the appropriate data-producing boundaries are stored as replay data and reused in later


Copyright © 2026, Oracle and/or its affiliates / Public

regression runs. Rather than requiring every environment to recreate the same business state, the test carries the context needed to reproduce the behavior, which can help make continuous integration (CI) and regression runs more reliable.

Using real data is not always the right approach. Representative success paths benefit from data captured in a live environment, but rare branches (e.g., a missing-rating path, an empty collection, or a specific error condition) are hard to reproduce naturally and may need a deliberately constructed data set. ATLAS uses captured data for representative paths and generated data for controlled conditions. It can also reduce large numbers of captured responses to the subset that preserves the behavior under test.

The essential requirement is consistency. If an upstream collection of 50 employees is reduced to 3, the downstream data must stay consistent with those 3. Identifiers, counts, parent-child relationships, LOOP inputs, and dependent fixtures must all describe the same scenario. Otherwise, the model is asked to reason over a scenario that is technically valid but logically contradictory.

Masking of recorded data applies configured policies to evidence while preserving the relationships needed for replay. Tests that require original attachments retain those files alongside their definitions. Live recording uploads the originals; file replay uses recorded reader output. Evidence masking does not mask the bytes in original attachments.

## Testing complex workflows without running every test end to end

Orchestration adds another challenge. A LOOP that processes 100 employees and runs several operations per employee poses a problem at both extremes. Recording and replaying every internal operation for every employee across every scenario produces an enormous suite, while mocking the LOOP entirely leaves its internal operations untested.

ATLAS combines the tests at two boundaries. One test captures the complete LOOP result and replays it at the boundary, allowing downstream behavior to be tested without re-running every iteration. Another test exercises the LOOP body with a single, consistent, representative item, so internal routing, transformations, model calls, and agent behavior run under controlled conditions. Together, these tests provide help teams evaluate both the LOOP's contract with the rest of the workflow and its internal behavior without multiplying the regression workload. The goal is to choose boundaries that keep the tests manageable while preserving the behavior that matters.

## A conversation is more than a sequence of answers

A workflow may retrieve information, retain a selection, answer a follow-up using that selection, and retrieve new information only when asked. Single-turn tests cannot prove continuity across those interactions. ATLAS conversation tests execute ordered messages using the same conversation and saved runtime state.

A useful test conversation can load a team, select an employee, ask about "that employee," refresh the data, clear the selection, and verify a new retrieval. Per-step fixtures allow the initial load and refresh to return different but consistent data. Assertions can verify paths, execution order, and counts as well as output and semantic quality.

## Suspension and resumption are observable contracts

Supported HUMAN interactions verify suspension and continuation through enabled approval, rejection, or feedback actions. Supported WAIT interactions verify timeout continuation without waiting for the configured wall-clock duration. The test checks that the expected execution was suspended and that the resume action continues from that state.


Copyright © 2026, Oracle and/or its affiliates / Public

These capabilities have defined boundaries. HUMAN support is limited to the CHAT channel. HUMAN and WAIT interactions inside nested jobs or LOOP, WHILE, or PARALLEL containers are outside the described support. Users can create and maintain custom tests for general follow-up and caching journeys, while workflow sync discovers supported HUMAN and WAIT interactions.

## Agentic applications need more than one definition of correct

Controlling the environment addresses half the problem. Teams must also decide whether the application behaved correctly, and agentic systems need more than one kind of judgment.

Some properties should be exact, such as a particular route executed, a prohibited branch was not taken, a required node ran, the response carried a particular structure. These are observable and should not depend on interpretation. ATLAS validates them with deterministic assertions. Generated content needs a different approach. Did the response answer the question, stay grounded in the supplied data, include the important facts, avoid unsupported conclusions, and offer a useful recommendation? Those criteria can be met even when the wording changes, so ATLAS complements deterministic assertions with semantic evaluation of generated output.

> ✓ **The principle**
>
> **Assert what must be exact. Evaluate what may legitimately vary.**

Structural correctness should not turn subjective just because AI is involved, and legitimate generative variation should not be treated as a defect just because two strings differ.

## Tests must evolve with the application

Agentic workflows change quickly. A developer edits a prompt, adds a condition, introduces a branch, restructures a LOOP, changes an application stage, or replaces a model. Each change can create a gap between the current application and the tests written for an earlier version. ATLAS treats this as a lifecycle problem. It analyzes the changed workflow, rediscovers meaningful scenarios, flags missing coverage, updates stale tests, and surfaces tests tied to paths that no longer exist. It does this without regenerating everything and discarding deliberate developer work.

This discipline matters most during repair. A system that responds to every failure by relaxing its expectations would eventually make every test pass and undermine the suite's value. ATLAS instead repairs the smallest appropriate element. If the fixture is stale, fix the fixture. If an assertion is needlessly brittle, adjust the assertion. If the application is wrong, fix the workflow rather than teaching the test to accept the defect. Lifecycle automation preserves the behavioral intent the suite represents as the application evolves.

## From regression testing to continuous AI evaluation

When scenarios and enterprise context can be reproduced reliably, the same infrastructure does more than catch regressions. It becomes a controlled environment for comparing models and configurations. Suppose a team wants to know whether a different model improves an Agentic Application. Running a few prompts manually gives an impression, but conclusions are hard to trust when the underlying data also changes between runs. With ATLAS, candidate models run against the same scenarios, inputs, replay data, workflow, and evaluation criteria, making the model the primary variable.


Copyright © 2026, Oracle and/or its affiliates / Public

Teams can evaluate how each model behaves across scenarios relevant to the application. The regression suite becomes an application-specific evaluation corpus, with the same testing infrastructure now supporting model comparison and optimization.

## The regression suite becomes an evaluation corpus

ATLAS supports manual comparisons and optimization sweeps. A node-level sweep helps evaluate suitable models for each model-configurable large language model (LLM) or agent node; a workflow-level sweep compares candidate models across eligible nodes. Classification, extraction, and business synthesis may justify different model assignments within one workflow.

Quality gates come first. A candidate must preserve accepted functional and semantic behavior before lower AI Unit usage, fewer tokens, or lower latency can support a recommendation. Fresh, labeled runs prevent old reports from being reused as current evidence. Reports identify accepted and rejected candidates and can validate a combined model placement plan.

Experimental overrides do not modify source code. A validated model placement is applied only within the authorized scope, followed by the required suite validation. Smaller replay data does not demonstrate that the production workflow uses less context or costs less to execute.

## Economic regression complements functional regression

A workflow can remain correct while consuming more AI Units. A longer prompt, an extra model call, or repeated retrieval may change the cost per outcome. Per-test budget assertions detect consumption beyond a configured limit. Economic regression testing connects representative outcomes with expected production volume, an approved allocation, and accepted consumption baselines.

An economic profile projects usage and utilization from selected outcomes and their execution volumes. Replay and live execution use separate baselines because replayed operations do not reproduce every live cost. Runtime metering is authoritative when available; compatibility estimates and unavailable measurements are identified rather than presented as fabricated precision.

Ordinary comparisons retain accepted baselines. Changes to profiles, allocations, volumes, and baselines require explicit approval. Initial economic setup measures representative replay and live behavior and presents assumptions before profile creation. A profile-specific policy can enforce standalone economic checks. A consolidated package runs report economic results separately as advisory evidence, preserving functional qualification totals.

Scenario generation, recording and replay, assertions, semantic evaluation, test repair, and model comparison are each useful on their own, but connecting them into a continuous engineering loop increases their value.


Copyright © 2026, Oracle and/or its affiliates / Public

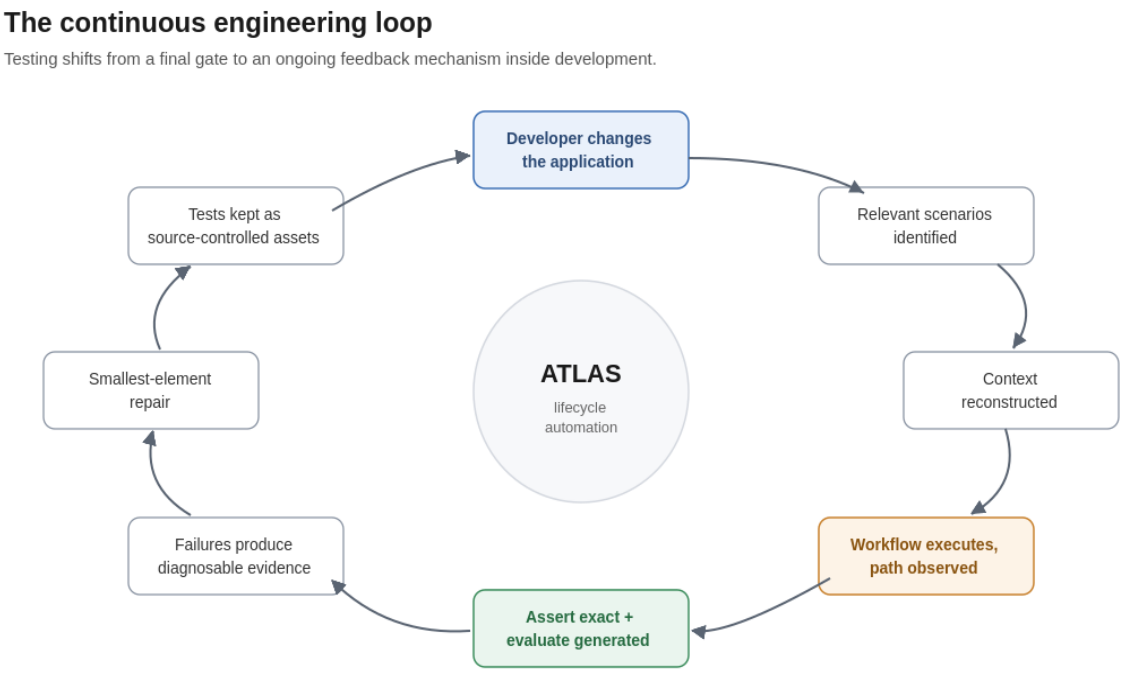


**Figure 2.** Connected capabilities form a continuous loop that keeps tests synchronized with an evolving application.

## ATLAS recommendations turns evidence into action

A passing suite demonstrates behavior, but it does not identify every potential improvement. When enabled for the workspace, ATLAS recommendations evaluate current workflow evidence and propose executable actions with a reason, expected impact, and validation plan.

This evidence combines source code and graph structure, data lineage, coverage, assertions, semantic judgments, effective model assignments, catalog information, AI Units, tokens, latency, and economic state. Recommendations can identify quality repairs, missing tests, repeated work, improvements to production context, model placement opportunities, and economic concerns.

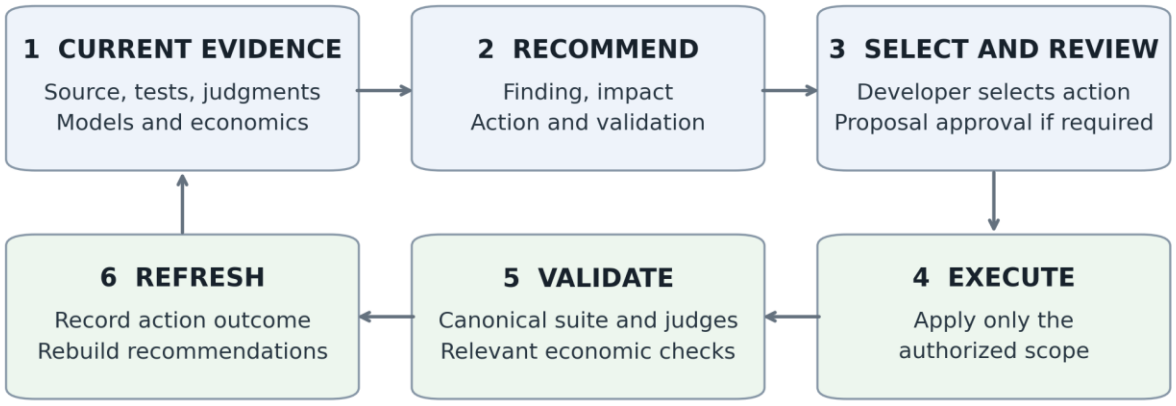


**Figure 3.** Current evidence produces recommendations. Selected and authorized work is executed, validated, and followed by refreshed actions.

The developer selects an action using natural language. Analysis or a sweep can proceed from that selection. A source code change designed by a model becomes a focused proposal for review; selecting an exact, validated model placement can authorize it. Protected allocation, profile, and baseline changes retain their own approval requirements.


Copyright © 2026, Oracle and/or its affiliates / Public

The model interprets and ranks opportunities. The command-line interface (CLI) checks evidence freshness, validates executable operations, enforces approval requirements, and produces authoritative results. Changes to relevant source code, contracts, models, reports, or economic state invalidate affected selections.

After execution and required canonical validation, ATLAS records the outcome and rebuilds recommendations. An unsuccessful validation remains evidence for repair or review. “ATLAS Optimum Achieved” is a qualified result for the current artifact and complete evidence; it is not permanent certification. Incomplete evidence or unresolved concerns remain visible.

## Making the testing lifecycle part of delivery

Source-controlled tests retain scenario intent, fixtures, assertions, semantic rubrics, tags, and required attachments. Reports provide generated evidence. Individual results are combined into workflow suites and consolidated qualification summaries, with links connecting each status to the observations behind it.

Continuous integration and continuous delivery (CI/CD) can run committed workflow tests against file replay and the configured remote judge. Failed assertions, execution errors, and required judgments left pending prevent successful functional qualification. Noninteractive runs execute the committed contracts; they do not initiate recommendation analysis or repair tests to manufacture success.

When a developer changes the application, relevant scenarios are identified and the required context is reconstructed. The workflow executes, and its path is observed. Deterministic behavior is validated, and generated behavior is evaluated semantically. Failures produce evidence for diagnosis and repair, but the resulting tests remain source-controlled assets, ready to run again as the application continues to change. Testing becomes an ongoing feedback mechanism in agentic development. ATLAS recommendations connect the evidence to selected improvements, while canonical validation and economic comparisons show whether those changes preserve accepted behavior and efficiency.

**Connect with us**

Call +**1.800.ORACLE1** or visit **oracle.com**. Outside North America, find your local office at: **oracle.com/contact**.

Copyright © 2026, Oracle and/or its affiliates. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle, Java, MySQL, and NetSuite are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.