Shawn C. Wright

AWO Workflow v4.2 — Test Plan v1.0

1. Purpose

This document defines the test plan for validating the Aurora Workflow Orchestration (AWO) GitHub Actions workflow prior to declaring workflow behavior **locked** (v4.2).

The goal is to verify:

- Environment capture
- Approval logic (including override behavior)
- Attestation independence
- Commit gating
- Logging and governance structure
- Basic integrity and ordering of generated artifacts

This plan is intentionally minimal but standards-grade: it focuses on the behaviors that matter for governance, provenance, and reproducibility.

---

2. System Under Test

- **Workflow file:** `.github/workflows/awo_run_v4.1.yml`
- **Trigger:** `workflow_dispatch`
- **Key input:** `allow_self_approval` ("0" or "1")
- **Primary outputs per run:**
    - `runs/run_YYYY-mm-ddTHH-MM-SSZ/` directory
    - `run_manifest.json`
    - `environment.json`
    - `provenance.json`
    - `gate_decision.yml`
    - `approval.json`
    - `SHA256SUMS.txt`
    - `ATTESTATION.txt` (+ `.sig`, `.cert`)
    - `artifacts/notes/*`

v4.2 refers to the **behavioral profile** of the workflow once this test plan is fully executed and all checks pass.

---

3. Test Phases

Testing is divided into two phases:

1. **Phase A — Canonical Behavior (Runs #1–#6)**
   Validate the normal and override paths under ideal conditions.

2. **Phase B — Governance & Integrity Edge Cases (Runs #7–#9)**
   Validate ordering, artifact validation, and multi-approval semantics.

All runs MUST be executed against the `main` branch of the repository unless explicitly noted otherwise.

---

4. Preconditions

Before executing any tests:

- ☐ Repository is clean (no uncommitted changes on `main`).
- ☐ `.github/workflows/awo_run_v4.1.yml` passes YAML syntax validation.
- ☐ Schemas in `/schemas/` are up to date.
- ☐ `root-sha256sums.yml` workflow is healthy.
- ☐ Doc Guard passes or has only known, documented warnings.

---

5. Phase A — Canonical Behavior (Runs #1–#6)

5.1 Test Set A — Self-Approval Blocked

Intent:
Validate that self-approval is blocked by default and that lack of approval prevents commit.

Run #1 — Normal Approval Required

- **allow_self_approval:** `0`
- **Orchestrator:** Account A
- **Approver:** Account B
- **Expected:**
  - Workflow pauses at approval gate until Account B approves.
  - `gate_decision.yml` records a normal, non-override approval.
  - `approval.json` shows Approver = Account B, Orchestrator = Account A.
  - Commit proceeds only after approval.
  - Governance logs record independence satisfied.
  - All required artifacts present and schema-valid.

Run #2 — No Approval Provided

- **allow_self_approval:** `0`
- **Orchestrator:** Account A
- **Approver:** none
- **Expected:**
  - Workflow halts at approval gate (no commit).
  - `gate_decision.yml` indicates missing approval / failure.
  - No changes are committed to the repo.
  - A failure state is recorded in `run_manifest.json` or equivalent.
  - Governance logs show run as incomplete / blocked.

5.2 Test Set B — Self-Approval Allowed (Override)

Intent:
Validate override path and logging when self-approval is explicitly allowed.

Run #3 — Self-Approval with Override

- **allow_self_approval:** `1`
- **Orchestrator:** Account A (also Approver)
- **Expected:**
  - Workflow allows Account A to approve their own run.
  - `gate_decision.yml` explicitly records override / self-approval.
  - Governance logs show independence **not** satisfied but override recorded.
  - Commit proceeds after self-approval.
  - All artifacts present and valid.

Run #4 — Override + Governance Inspection

- **allow_self_approval:** `1`
- **Orchestrator:** Account A
- **Approver:** Account A
- **Expected:**
  - Same behavior as Run #3, but focus on:
    - Governance log entry in `governance/logs/`
    - Correct run ID cross-references
    - Accurate timestamps and actor identifiers
  - Confirm that this run is clearly distinguishable from non-override runs.

5.3 Test Set C — Cross-Account Orchestration

Intent:
Validate multi-agent governance with independent orchestrator and approver.

Run #5 — Orchestrator A, Approver B

- **allow_self_approval:** `0`
- **Orchestrator:** Account A
- **Approver:** Account B
- **Expected:**
  - Gate waits for Account B approval.
  - `approval.json` and `gate_decision.yml` show distinct orchestrator/approver.
  - Governance logs indicate independence satisfied.
  - Commit occurs after approval.
  - All artifacts present, including `environment.json` and `provenance.json`.

Run #6 — Same Roles, Small Repo Change

- **allow_self_approval:** `0`
- **Orchestrator:** Account A

- **Approver:** Account B
- **Repo change:** Small, deliberate modification (e.g., README typo fix).
- **Expected:**
  - Same behavior as Run #5.
  - `run_manifest.json` and `SHA256SUMS.txt` reflect the change.
  - Non-repudiation validated: it is clear which run produced which change.

---

6. Phase B — Governance & Integrity Edge Cases (Runs #7–#9)

6.1 Run #7 — Early Approval (Ordering Test)

Intent:
Ensure approval cannot “race ahead” of manifest finalization.

- **Scenario:** Approver attempts to approve before workflow produces final manifest/artifacts.
- **Expected:**
  - Either the approval is queued until a safe point, or the system rejects premature approval.
  - No commit is allowed until artifacts are generated and validated.
  - `gate_decision.yml` and logs reflect the actual ordering.

Implementation detail may depend on how the manual approval step is wired; the key is that ordering cannot produce an inconsistent run state.

6.2 Run #8 — Invalid Artifact (Schema/Integrity Test)

Intent:
Verify that malformed or incomplete artifacts cause a clean, logged failure.

- **Scenario:** Introduce a controlled error, e.g.:
  - Manually break a field in a generated JSON
  - Or simulate missing `environment.json`
- **Expected:**
  - Workflow fails or halts at validation.
  - No commit is pushed.
  - `run_manifest.json` or logs indicate validation failure.
  - Governance logs do not show a successful run.
  - Doc Guard and/or schema validation should flag the error.

The specific mechanism (manual edit vs. injected failure) should be documented in the run notes.

6.3 Run #9 — Duplicate Approval (Multi-Approval Semantics)

Intent:
Validate that multiple approvals do not corrupt governance state.

- **Scenario:**
  - Orchestrator: Account A
  - Approver 1: Account B
  - Approver 2: Account C (secondary, possibly redundant approver)
- **Expected:**
  - System either:
    - Accepts the first approval and ignores subsequent approvals, or
    - Records all approvals but treats them as non-conflicting.
  - No double-commit or extra runs are triggered.
  - Governance log clearly shows which approval satisfied the gate.

---

7. Post-Run Checks

After all runs (#1–#9) are complete:

- ☐ Confirm each run directory exists under `/runs/`.
- ☐ Validate presence and schema-conformance of:
  - `run_manifest.json`
  - `environment.json`
  - `provenance.json`
  - `gate_decision.yml`
  - `approval.json`
  - `SHA256SUMS.txt`
- ☐ Inspect `governance/logs/` for:
  - Independence checks
  - Overrides
  - Cross-account approvals
- ☐ Run Doc Guard and ensure no new critical warnings.
- ☐ Run root SHA256SUMS workflow and confirm integrity.

If all checks pass, the workflow behavior may be declared **locked** as v4.2.

---

8. Lock Criteria for Workflow v4.2

The AWO workflow may be considered behaviorally stable (v4.2) when:

- All nine runs complete with expected behavior.
- No unauthorized commit occurs.
- All governance-relevant events are logged and attributable.
- All required artifacts are present and schema-valid.
- No unexplained failures are observed in CI.

At that point, the workflow file SHOULD be: - Tagged in version control. - Referenced in the Architecture and Lifecycle specifications. - Treated as the behavioral baseline for future revisions.

---

9. Changelog

- **v1.0 (2025-11-24):** Initial test plan created for AWO workflow v4.2 lock.