AI Automation / Workflow Systems

Plan the migration from legacy workflows to new systems with rollback strategies — enables safe system replacement.
Difficulty: Advanced
Model: GPT-4 / Claude / Gemini
Use Case: System Migration, Legacy Replacement
Updated: May 2026
Why This Prompt Exists
Replacing legacy workflows is risky. One mistake breaks downstream systems, and without a rollback plan, you can’t recover quickly.

You get:

- migrations that break production (no staging environment test)
- no rollback plan (hours to revert, not minutes)
- data loss during migration (no verification step)
- parallel runs that drift (legacy and new systems diverge)
- no cutover window (migration drags on for weeks)

But migrations can be planned:

- discovery: what workflows are being replaced?
- dependency mapping: what relies on these workflows?
- parallel run: run both systems, compare outputs
- cutover: switch traffic atomically
- rollback: revert to legacy if issues arise

Without planning, migrations fail catastrophically.

This prompt designs safe workflow migration plans.

The Prompt

```
Assume the role of a migration architect who plans workflow
replacements.

Your task is to create a safe migration plan from legacy to new
workflows.

Generate:

1. MIGRATION SCOPE
   - Legacy workflows being replaced: [list]
   - New workflows replacing them: [list]
   - Reason for migration: [performance / maintainability / compliance
/ new features]

2. DEPENDENCY IMPACT ANALYSIS (from WS-02)
   - Downstream workflows that will be affected: [list]
   - Upstream workflows that feed into legacy: [list]
   - Shared resources (databases, APIs) in scope: [list]

3. MIGRATION STRATEGY

| Strategy | Description | Risk | Duration |
|----------|-------------|------|----------|
| Parallel run | Run both systems, compare outputs | Low | Weeks |
| Phased cutover | Migrate one workflow at a time | Medium | Days |
```

| Big bang | Cut over all at once | High | Hours |

4. PARALLEL RUN PLAN (recommended for high-risk)
   - Step 1: Run new workflow alongside legacy (no output)
   - Step 2: Compare outputs for [X] days
   - Step 3: Fix discrepancies
   - Step 4: Switch to new workflow for [X]% of traffic
   - Step 5: Gradually increase to 100%

5. CUTOVER PLAN
   - Cutover window: [date and time]
   - Expected downtime: [X minutes/seconds]
   - Communication plan: [who to notify, when]

6. ROLLBACK STRATEGY
   - Trigger conditions: [error rate > X% / data discrepancy > Y% / timeout]
   - Rollback steps: [revert configuration, restart legacy workflows]
   - Rollback time: [X minutes]
   - Data reconciliation: [how to handle data processed during rollback]

7. VALIDATION CHECKLIST
   - [ ] Legacy and new outputs identical for 1,000 test cases
   - [ ] Downstream workflows receive same data format
   - [ ] Performance meets or exceeds legacy
   - [ ] Rollback tested in staging

INPUTS:

```
Legacy workflows (from WS-01):
[PASTE LEGACY WORKFLOW DESCRIPTIONS]

New workflows:
[PASTE NEW WORKFLOW DESCRIPTIONS]

Dependency map (from WS-02):
[PASTE DEPENDENCIES]

Risk tolerance:
[HIGH (can tolerate brief failures) / MEDIUM / LOW (zero downtime
required)]

RULES:
- Parallel run first for critical workflows (build confidence)
- Test rollback in staging before production (always)
- Communicate cutover window to all downstream consumers
- Monitor both systems during parallel run (log discrepancies)
- Have a rollback trigger threshold (don't decide in the moment)
- Data reconciliation is harder than code rollback (plan for it)
```

How To Use It

- Parallel run first for critical workflows — builds confidence before cutover.
- Test rollback in staging before production — always.
- Communicate cutover window to all downstream consumers — surprises break things.
- Monitor both systems during parallel run — log every discrepancy.
- Have a rollback trigger threshold — don’t decide in the moment.
- Data reconciliation is harder than code rollback — plan for it.

Example Input

Legacy workflows:
“WF-001 (Legacy Lead Capture): Typeform → Salesforce via Zapier. WF-002 (Legacy Lead Scoring): Custom Python script on EC2.”

New workflows:
“WF-001-NEW: Typeform → Salesforce via API (direct). WF-002-NEW: Lead scoring in Make.com with error handling.”

Dependency map:
“WF-002 depends on data from WF-001. WF-003 (Routing) depends on WF-002.”

Risk tolerance:
“LOW — zero downtime, zero data loss”

Why It Works
Most migrations are planned as a single event — “we’ll switch on Tuesday” — with no rollback, no parallel run, and no confidence building.

This framework improves outcomes by forcing:

- dependency impact analysis (what breaks if this migration fails?)
- migration strategy selection (parallel, phased, or big bang)
- parallel run plan (compare outputs, build confidence)
- cutover planning (when, how, who to notify)
- rollback strategy (how to recover quickly)

Failure modes this prevents:

- Untested rollback — “we’ll figure it out if something goes wrong” (you won’t)
- No parallel run — discover discrepancies after cutover (too late)

- Data loss during migration — no reconciliation plan
- Downstream surprise — workflow changes format, consumers break

This improves on: “Big bang” migrations that hope for the best. Safe migrations are planned, tested, and reversible.

Related to: WS-02 (Dependencies) for impact analysis; WS-04 (Optimizer) for performance targets.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, debugging frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

See also Workflow Audit Trail Designer