RTMX

Patterns and Anti-Patterns

Best Practices for Requirements Traceability with RTMX

Author
RTMX Engineering

Version
1.0

Date
February 10, 2026

rtmx.ai • dev@rtmx.ai

Introduction

This guide defines recommended patterns and anti-patterns for working with RTMX. Following these patterns ensures requirements remain traceable, verifiable, and trustworthy.

Whether you're a developer integrating RTMX into your workflow, a team lead establishing processes, or an AI agent working in an RTMX-enabled project, these patterns will help you get the most value from requirements traceability.

i Canonical Source

This whitepaper expands on the patterns defined in `docs/patterns.md` in the RTMX repository. That file serves as the single source of truth and is designed for insertion into project CLAUDE.md files.

Core Principle: Closed-Loop Verification

RTMX is built on a fundamental principle:

Requirement status must be derived from evidence, not opinion.

This is **closed-loop verification**: tests determine status, and status flows back to inform what needs work.

```text
┌─────────────────────────────────────────────────────────┐
│                  CLOSED-LOOP VERIFICATION                │
│                                                         │
│   Requirements ──→ Tests ──→ Execution ──→ Status Update   │
│        ↑                                   │             │
│        └───────────────────────────────────┘             │
│                                                         │
│   Status reflects reality. Tests are the arbiter of truth. │
└─────────────────────────────────────────────────────────┘
```

Why Closed-Loop Matters

When status is derived from tests:

- **Releases are trustworthy** — 100% complete means all tests pass
- **Regressions are detected** — Failed tests automatically downgrade status
- **AI agents can't lie** — They can't claim completion without passing tests
- **Progress is auditable** — Git history shows exactly what changed and when

Verification Patterns

Pattern: Automated Status Updates

✓ **Pattern**

Use `rtmx verify --update` to derive status from test results

✗ **Anti-Pattern**

Manually editing the `status` field in rtm_database.csv

The `verify` command is the core of RTMX:

```bash
# In CI/CD pipeline
rtmx verify --update

# Local development (preview first)
rtmx verify --dry-run
rtmx verify --update
```

How It Works

1. Runs all tests with `@pytest.mark.req()` markers
2. Maps test outcomes to requirements
3. Updates status based on pass/fail results

Test Result	Status Transition
All pass	→ COMPLETE
Some pass, none fail	→ PARTIAL
Any fail	COMPLETE → PARTIAL (regression)
No tests	Status unchanged

✓ **Key Insight**

Status reflects what the code **actually does**, not what someone **hopes** it does.

Anti-Pattern: Manual Status Edits

✗ **Never Do This**

Manually editing the `status` field breaks the verification loop and makes the RTM untrustworthy.

Symptoms of this anti-pattern:

- Status says COMPLETE but tests fail

- No test coverage for "complete" requirements
- Status changes without corresponding code changes
- Requirements marked complete before implementation

Why it's harmful: Manual status updates transform the RTM from a verification record into a wish list. When you can't trust status, you can't trust release readiness.

Pattern: Test-Linked Requirements

Every requirement should have at least one test with `@pytest.mark.req()`:

```python
@pytest.mark.req("REQ-AUTH-001")
@pytest.mark.scope_unit
@pytest.mark.technique_nominal
@pytest.mark.env_simulation
def test_user_can_login():
    """Verify REQ-AUTH-001: User can log in."""
    user = create_test_user()
    result = login(user.email, user.password)
    assert result.success
    assert result.session_token is not None
```

Sync test metadata to the RTM:

```bash
rtmx from-tests --update
```

This populates `test_module` and `test_function` columns, creating **bidirectional traceability**.

Anti-Pattern: Orphan Tests

✓ Pattern	✗ Anti-Pattern
Add `@pytest.mark.req()` to every test	Write tests without requirement markers

Detection:

```bash
rtmx from-tests --missing  # Shows unlinked tests
```

Orphan tests provide no evidence for requirement completion. They may test important functionality, but that functionality isn't tracked.

Development Workflow Patterns

Pattern: Spec-First Development

Write the requirement specification **before** writing code:

Step	Action
1	Define requirement in RTM database
2	Create specification file
3	Write acceptance criteria
4	Write failing tests
5	Implement to pass tests
6	Run `rtmx verify --update`

Anti-Pattern: Code-First, Spec-Never

✓ Pattern

Define requirement ⟶ Write spec ⟶ Write test ⟶ Implement ⟶ Verify

✗ Anti-Pattern

Write code ⟶ Maybe write tests ⟶ Never create requirement

Features without requirements can't be verified, prioritized, or traced. When asked "what does the system do?", the answer becomes "read the code."

Pattern: Phase Gates in CI

Block releases until phase requirements are verified:

```yaml
# .github/workflows/release.yml
- name: Verify Phase Requirements
  run: |
    rtmx verify --update
    rtmx status --json | jq -e '.phases["1"].complete == true'
```

Anti-Pattern: Phase as Suggestion

⚠️ Don't Skip Phase Gates

Releasing with incomplete phases means shipping unverified functionality. Phases exist to ensure quality gates, not as optional guidance.

Agent Integration Patterns

Pattern: Agent as Implementer, RTMX as Verifier

Agent Workflow:

1. Read requirement spec from `docs/requirements/`
2. Write tests with `@pytest.mark.req()`
3. Implement code to pass tests
4. Run `rtmx verify --update`
5. Commit (status already updated by verification)

The key insight: agents **implement**, RTMX **verifies**. Agents never determine status—tests do.

Anti-Pattern: Agent Status Claims

✓ Pattern

```
subprocess.run(["rtmx", "verify", "--update"])
```

Evidence-based status from tests

✗ Anti-Pattern

```
db.update("REQ-XXX", status=Status.COMPLETE)
```

Opinion-based status claim

Agent opinions about completion are unreliable. Tests may not exist, may fail, or may not cover the requirement. Only `rtmx verify` provides evidence-based status.

Pattern: RTM as Development Contract

Agents should read the RTM to understand what to build:

```bash
# Discover next task
rtmx backlog --phase 2 --limit 1

# Read specification
cat docs/requirements/CATEGORY/REQ-XXX.md

# Check dependencies
rtmx deps --req REQ-XXX

# Implement, then verify
rtmx verify --update
```

The RTM provides:

- **What to build** — requirement text
- **How to verify** — linked tests
- **What's blocking** — dependencies
- **Priority order** — phase, priority

Anti-Pattern: Ignoring Dependencies

Implementing requirements before their dependencies are complete often requires rework. Check `blockedBy` fields before starting work.

CI/CD Patterns

Pattern: Verify on Every PR

```
yaml

# .github/workflows/ci.yml
jobs:
  verify:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install RTMX
        run: pip install rtmx
      - name: Verify Requirements
        run: rtmx verify --update
      - name: Check for Regressions
        run: |
          git diff --exit-code docs/rtm_database.csv || \
            echo "::warning::RTM status changed"
```

Pattern: RTM Diff in PRs

Show requirement status changes in pull requests:

```
yaml

- name: RTM Diff
  run: |
    git fetch origin main
    rtmx diff origin/main HEAD --format markdown >> $GITHUB_STEP_SUMMARY
```

This surfaces new requirements, completions, and regressions.

Quick Reference

Commands

Command	Purpose	When
`rtmx verify --dry-run`	Preview status changes	Before updating
`rtmx verify --update`	Update status from tests	After implementation
`rtmx from-tests --update`	Sync test metadata	After adding tests
`rtmx health`	Validate RTM integrity	Before releases

Status Sources

Source	Trustworthy?	Notes
`rtmx verify --update`	Yes	Evidence-based

Manual CSV edit	No	Opinion-based
Agent claim	No	Unverified
CI pipeline	Yes	Automated verification

Verification Checklist

Before claiming a requirement is complete:

- Test exists with `@pytest.mark.req("REQ-XXX")`
- Test passes locally
- `rtmx verify --update` shows COMPLETE
- CI pipeline passes
- Specification acceptance criteria met

Summary

Do This (Pattern)	Not This (Anti-Pattern)
`rtmx verify --update`	Manual status edits
`@pytest.mark.req()` on all tests	Orphan tests
Spec-first development	Code-first, spec-never
Phase gates in CI	Phase as suggestion
Agent implements, RTMX verifies	Agent claims completion
Respect dependencies	Ignore blockedBy

✓ Remember

The RTM is a **verification record**, not a wish list.
Status must be **earned** through passing tests, not **claimed** through manual edits.

For more information, visit **rtmx.ai**
Questions? Contact **dev@rtmx.ai**