CANONICAL LIVING OVERVIEW GUIDE

Agentic Systems, Harnesses, Tools, Memory, and Multi-Agent Orchestration

A comprehensive guide to designing, securing, operating, evaluating, and scaling agent runtimes, harness fabrics, tools, skills, MCP integrations, memory, context, planning, multi-agent teams, multi-model parallelism, governance, evidence, and recovery.

PERMANENT ID
MYTHOS-GUIDE-AGENT-0001

PUBLICATION CLASS
FIELD GUIDE

VERSION / STATUS
1.0.0-candidate / CANDIDATE

PERMANENT PUBLICATION IDENTITY

Agentic Systems, Harnesses, Tools, Memory, and Multi-Agent Orchestration

Document ID
MYTHOS-GUIDE-AGENT-0001

Version
1.0.0-candidate

Status
Candidate Living Guide

Review date
2026-10-26

Publication position
This is a living field guide. It is designed for frequent revision while retaining a stable permanent identity. Candidate status means the content is ready for structured use and review but is not represented as an external certification or authority publication.

LEVEL	CLASS	CADENCE	EVIDENCE
L1-L4	FIELD GUIDE	QUARTERLY	SOURCE-BOUND
Foundational through frontier	Practical and educational	Interim updates as needed	Limitations remain visible

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-SWE; MYTHOS-DAT; MYTHOS-SEC; MYTHOS-CLD
Audience	AI platform engineers, agent-framework developers, software architects, security engineers, automation teams, model evaluators, and technical leaders.
Prerequisites	Software architecture, APIs, testing, basic AI/LLM concepts, and security fundamentals. Later chapters benefit from distributed-systems and observability experience.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

Distinguish agents, workflows, harnesses, tools, skills, models, memory, and orchestration responsibilities.

Design bounded agent execution with explicit authority, identity, evidence, budgets, and recovery.

Engineer multi-agent and multi-model parallelism without losing determinism, provenance, or human control.

Evaluate agent systems through task, trajectory, tool, security, cost, latency, and operational measures.

Scale agentic applications using portable contracts, observable runtimes, and controlled capability fabrics.

How to use this guide

Read the guide as a progressive system rather than a disconnected encyclopedia. Foundational material establishes the mental model; practitioner sections convert it into repeatable implementation and operations; advanced sections expose architecture, scale, security, and failure behavior; frontier sections describe technologies whose evidence or maturity is still evolving.

Suggested reading path
New practitioners should follow the chapters in order and complete the laboratories. Experienced engineers may begin with the architecture, failure, validation, or frontier sections, then use the related-document map to deepen a specific topic.

Learning and assurance model

Level	Reader outcome
L1 - Foundational	Terminology, first principles, component roles, packet or state flow, and simple working examples.
L2 - Practitioner	Implementation, configuration, automation, testing, troubleshooting, and operational evidence.
L3 - Advanced	Architecture, scale, concurrency, resilience, threat models, performance, and complex migrations.
L4 - Frontier	Emerging systems, research directions, technical uncertainty, readiness gates, and adoption signals.

Contents

CHAPTER 01

Agentic systems mental model and operating boundaries

Define an agent as a governed system that observes, plans, acts, evaluates, and terminates within explicit authority rather than as a model with a loop.

Chapter operating position

Define an agent as a governed system that observes, plans, acts, evaluates, and terminates within explicit authority rather than as a model with a loop.

1.1 Agents, workflows, and autonomous control loops

Agents, workflows, and autonomous control loops is treated here as an engineering capability rather than a vocabulary item. Separate deterministic workflows, model-assisted steps, adaptive agents, and open-ended autonomy by authority and failure consequences. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for agents, workflows, and autonomous control loops.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating agents, workflows, and autonomous control loops as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for agents, workflows, and autonomous control loops.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

1.2 Intent, plans, actions, observations, and outcomes

Intent, plans, actions, observations, and outcomes is treated here as an engineering capability rather than a vocabulary item. Represent goals, decompositions, tool actions, environment observations, verification, and final outcomes as durable typed records. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for intent, plans, actions, observations, and outcomes.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating intent, plans, actions, observations, and outcomes as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for intent, plans, actions, observations, and outcomes.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

1.3 Control plane, execution plane, and evidence plane

Control plane, execution plane, and evidence plane is treated here as an engineering capability rather than a vocabulary item. Separate policy and scheduling from effectful execution and from immutable evidence used to explain what occurred. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for control plane, execution plane, and evidence plane.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating control plane, execution plane, and evidence plane as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for control plane, execution plane, and evidence plane.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

1.4 Autonomy levels and human authority

Autonomy levels and human authority is treated here as an engineering capability rather than a vocabulary item. Define escalation, approval, stop, rollback, and forbidden-action boundaries for each mission and environment. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for autonomy levels and human authority.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating autonomy levels and human authority as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for autonomy levels and human authority.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 01 laboratory and review

Practical laboratory

Model a repository-change mission as a state machine with approval gates, tool permissions, failure transitions, evidence records, and a deterministic termination condition.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Extend the model to long-running autonomous operations, physical systems, or cross-organizational agents and document where present governance techniques become insufficient.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 02

Harnesses, runtimes, and lifecycle contracts

Build a runtime substrate that can host multiple agent frameworks and models without allowing framework-specific behavior to become the system of record.

Chapter operating position
Build a runtime substrate that can host multiple agent frameworks and models without allowing framework-specific behavior to become the system of record.

2.1 Harness responsibilities and portability

Harness responsibilities and portability is treated here as an engineering capability rather than a vocabulary item. Define adapters for prompts, tools, state, events, checkpoints, cancellation, budgets, and evidence across heterogeneous harnesses. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for harness responsibilities and portability.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating harness responsibilities and portability as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for harness responsibilities and portability.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

2.2 Mission lifecycle and durable execution

Mission lifecycle and durable execution is treated here as an engineering capability rather than a vocabulary item. Implement admission, initialization, execution, suspension, resume, retry, compensation, completion, and archival semantics. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for mission lifecycle and durable execution.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating mission lifecycle and durable execution as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for mission lifecycle and durable execution.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

2.3 Isolation, sandboxes, and resource governance

Isolation, sandboxes, and resource governance is treated here as an engineering capability rather than a vocabulary item. Constrain filesystem, process, network, credential, model, GPU, memory, and time access by mission and agent identity. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for isolation, sandboxes, and resource governance.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating isolation, sandboxes, and resource governance as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for isolation, sandboxes, and resource governance.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

2.4 Checkpoints, replay, and rollback

Checkpoints, replay, and rollback is treated here as an engineering capability rather than a vocabulary item. Capture enough state to reproduce decisions, resume safely, and reverse effects without duplicating irreversible actions. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for checkpoints, replay, and rollback.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating checkpoints, replay, and rollback as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for checkpoints, replay, and rollback.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 02 laboratory and review

Practical laboratory

Implement a minimal harness adapter that runs the same governed mission through two different agent frameworks while preserving one lifecycle and evidence schema.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate multi-harness fabrics in which specialized runtimes are selected dynamically by task type, risk, hardware, and model availability.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 03

Tools, skills, MCP, plugins, and capability security

Treat external capabilities as security-sensitive contracts with explicit schemas, identities, permissions, provenance, and revocation.

Chapter operating position

Treat external capabilities as security-sensitive contracts with explicit schemas, identities, permissions, provenance, and revocation.

3.1 Tool contracts and effect classification

Tool contracts and effect classification is treated here as an engineering capability rather than a vocabulary item. Classify read, write, execute, communicate, spend, deploy, and destructive effects and bind each to authorization and evidence. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for tool contracts and effect classification.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating tool contracts and effect classification as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for tool contracts and effect classification.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

3.2 MCP lifecycle, transports, resources, prompts, and tools

MCP lifecycle, transports, resources, prompts, and tools is treated here as an engineering capability rather than a vocabulary item. Apply capability negotiation, initialization, transport security, schema validation, and lifecycle management to MCP clients and servers. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for mcp lifecycle, transports, resources, prompts, and tools.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating mcp lifecycle, transports, resources, prompts, and tools as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for mcp lifecycle, transports, resources, prompts, and tools.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

3.3 Skills, plugins, and extension packaging

Skills, plugins, and extension packaging is treated here as an engineering capability rather than a vocabulary item. Package reusable capability logic with versions, dependencies, permissions, tests, provenance, and compatibility declarations. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for skills, plugins, and extension packaging.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating skills, plugins, and extension packaging as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for skills, plugins, and extension packaging.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

3.4 Tool-result trust and prompt-injection containment

Tool-result trust and prompt-injection containment is treated here as an engineering capability rather than a vocabulary item. Treat tool output and retrieved content as untrusted data, preserve origin, sanitize control channels, and prevent authority confusion. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency,

orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for tool-result trust and prompt-injection containment.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating tool-result trust and prompt-injection containment as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for tool-result trust and prompt-injection containment.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 03 laboratory and review

Practical laboratory
Create a mock MCP server with one read-only and one effectful tool. Add permissions, user approval, schema validation, timeouts, audit evidence, and prompt-injection tests.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore capability proofs, confidential tool execution, signed tool manifests, and decentralized trust without assuming these techniques are mature.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 04

Context, memory, knowledge, and temporal continuity

Engineer the information available to agents as a governed resource with source authority, relevance, chronology, privacy, and token-cost constraints.

Chapter operating position

Engineer the information available to agents as a governed resource with source authority, relevance, chronology, privacy, and token-cost constraints.

4.1 Context assembly and budget allocation

Context assembly and budget allocation is treated here as an engineering capability rather than a vocabulary item. Select instructions, repository state, task history, evidence, tools, and domain knowledge within explicit token and latency budgets. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for context assembly and budget allocation.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating context assembly and budget allocation as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for context assembly and budget allocation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

4.2 Working, episodic, semantic, and procedural memory

Working, episodic, semantic, and procedural memory is treated here as an engineering capability rather than a vocabulary item. Separate transient reasoning state, mission episodes, durable knowledge, and reusable procedures by lifecycle and access policy. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for working, episodic, semantic, and procedural memory.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating working, episodic, semantic, and procedural memory as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for working, episodic, semantic, and procedural memory.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

4.3 Retrieval, grounding, contradiction, and freshness

Retrieval, grounding, contradiction, and freshness is treated here as an engineering capability rather than a vocabulary item. Rank sources, preserve citations, detect conflicts, track versions, and prevent stale or low-authority memory from silently controlling action. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for retrieval, grounding, contradiction, and freshness.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating retrieval, grounding, contradiction, and freshness as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for retrieval, grounding, contradiction, and freshness.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

4.4 Trajectory compaction and temporal integrity

Trajectory compaction and temporal integrity is treated here as an engineering capability rather than a vocabulary item. Compress long histories while preserving decisions, unresolved risks, causal order, approvals, and evidence needed for continuation. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for trajectory compaction and temporal integrity.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating trajectory compaction and temporal integrity as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for trajectory compaction and temporal integrity.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 04 laboratory and review

Practical laboratory
Run a long task that exceeds the model context window. Build a compaction artifact, resume in a new session, and demonstrate that decisions and unresolved risks remain intact.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Investigate memory-native model architectures, continual adaptation, semantic caches, and private on-device memory with measurable forgetting and contamination controls.

Review gate
Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 05

Planning, decomposition, scheduling, and parallelism

Turn goals into bounded work graphs whose dependencies, concurrency, budgets, and completion criteria remain visible and testable.

Chapter operating position
Turn goals into bounded work graphs whose dependencies, concurrency, budgets, and completion criteria remain visible and testable.

5.1 Plan representations and task graphs

Plan representations and task graphs is treated here as an engineering capability rather than a vocabulary item. Use typed tasks, dependencies, preconditions, expected artifacts, verification methods, and termination criteria instead of prose-only plans. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for plan representations and task graphs.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating plan representations and task graphs as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for plan representations and task graphs.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

5.2 Supervisor, specialist, reviewer, and verifier roles

Supervisor, specialist, reviewer, and verifier roles is treated here as an engineering capability rather than a vocabulary item. Allocate planning, implementation, domain expertise, security review, testing, and synthesis to agents with distinct authority. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for supervisor, specialist, reviewer, and verifier roles.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating supervisor, specialist, reviewer, and verifier roles as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for supervisor, specialist, reviewer, and verifier roles.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

5.3 Fan-out, fan-in, map-reduce, and speculative execution

Fan-out, fan-in, map-reduce, and speculative execution is treated here as an engineering capability rather than a vocabulary item. Parallelize independent work, compare alternatives, cancel losing branches, and merge only evidence-supported results. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for fan-out, fan-in, map-reduce, and speculative execution.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating fan-out, fan-in, map-reduce, and speculative execution as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for fan-out, fan-in, map-reduce, and speculative execution.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

5.4 Resource scheduling and workstation saturation

Resource scheduling and workstation saturation is treated here as an engineering capability rather than a vocabulary item. Schedule models, CPUs, GPUs, memory, storage, and external rate limits while avoiding contention and priority inversion. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for resource scheduling and workstation saturation.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating resource scheduling and workstation saturation as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for resource scheduling and workstation saturation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 05 laboratory and review

Practical laboratory

Execute a four-agent software task with planner, two parallel implementers, and verifier. Measure critical path, duplicated work, merge conflicts, token use, and result quality.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore hundreds of concurrent micro-agents, heterogeneous local/cloud model arrays, and adaptive scheduling while retaining a single verified outcome.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 06

Evaluation, observability, and evidence engineering

Measure agent systems at task, trajectory, decision, tool, system, safety, cost, and operator levels.

Chapter operating position
Measure agent systems at task, trajectory, decision, tool, system, safety, cost, and operator levels.

6.1 Task and outcome evaluation

Task and outcome evaluation is treated here as an engineering capability rather than a vocabulary item. Define success, partial success, unacceptable failure, environment state, and evidence before executing the mission. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for task and outcome evaluation.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating task and outcome evaluation as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for task and outcome evaluation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

6.2 Trajectory and decision evaluation

Trajectory and decision evaluation is treated here as an engineering capability rather than a vocabulary item. Score planning quality, unnecessary steps, recovery, unsupported assumptions, tool selection, and policy adherence. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for trajectory and decision evaluation.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating trajectory and decision evaluation as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for trajectory and decision evaluation.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

6.3 Agent and tool telemetry

Agent and tool telemetry is treated here as an engineering capability rather than a vocabulary item. Trace model calls, prompts, tool invocations, memory access, approvals, retries, costs, latency, and final artifacts with correlation IDs. The design objective is

to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for agent and tool telemetry.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating agent and tool telemetry as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for agent and tool telemetry.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

6.4 Evidence bundles and reproducibility

Evidence bundles and reproducibility is treated here as an engineering capability rather than a vocabulary item. Package inputs, versions, plans, actions, logs, diffs, tests, approvals, and results so another engineer can audit the mission. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

Define the authoritative source and ownership boundary for evidence bundles and reproducibility.

Make preconditions, invariants, error states, and recovery actions explicit.

Instrument the critical path with stable identifiers, timestamps, and outcome codes.

Validate in a representative environment before widening blast radius.

Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

Treating evidence bundles and reproducibility as a product feature rather than an end-to-end engineering behavior.

Automating an undocumented process and scaling its ambiguity.

Relying on a happy-path demonstration without degraded-state or rollback testing.

Using aggregate dashboards without retaining trace-level evidence.

Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for evidence bundles and reproducibility.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 06 laboratory and review

Practical laboratory

Instrument a tool-using agent with OpenTelemetry-compatible traces and generate an evidence bundle that reconstructs a failed and a successful mission.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Evaluate privacy-preserving telemetry, learned evaluators, causal attribution, and continuous production benchmarks without allowing opaque judges to become sole authority.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 07

Security, governance, and failure containment

Prevent agent systems from converting model uncertainty or hostile input into uncontrolled authority, data exposure, or irreversible action.

Chapter operating position

Prevent agent systems from converting model uncertainty or hostile input into uncontrolled authority, data exposure, or irreversible action.

7.1 Identity, authentication, and delegated authority

Identity, authentication, and delegated authority is treated here as an engineering capability rather than a vocabulary item. Assign durable identities to users, agents, services, tools, and missions and issue narrow, expiring, revocable capabilities. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for identity, authentication, and delegated authority.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating identity, authentication, and delegated authority as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for identity, authentication, and delegated authority.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

7.2 Agentic threat modeling

Agentic threat modeling is treated here as an engineering capability rather than a vocabulary item. Address prompt injection, tool misuse, memory poisoning, confused deputy, excessive agency, credential theft, cascading actions, and unsafe self-modification. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for agentic threat modeling.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating agentic threat modeling as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for agentic threat modeling.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

7.3 Policy, approvals, separation of duties, and budgets

Policy, approvals, separation of duties, and budgets is treated here as an engineering capability rather than a vocabulary item. Bind risk classes to human approval, dual control, spend limits, data boundaries, and independent verification. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for policy, approvals, separation of duties, and budgets.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating policy, approvals, separation of duties, and budgets as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for policy, approvals, separation of duties, and budgets.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

7.4 Incident response, kill switches, and recovery

Incident response, kill switches, and recovery is treated here as an engineering capability rather than a vocabulary item. Detect unsafe behavior, stop execution, revoke credentials, contain effects, preserve evidence, restore state, and learn from failure. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for incident response, kill switches, and recovery.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating incident response, kill switches, and recovery as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for incident response, kill switches, and recovery.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 07 laboratory and review

Practical laboratory

Red-team an agent with malicious retrieved content, poisoned memory, ambiguous authority, and an unsafe tool. Demonstrate prevention, detection, containment, and evidence.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Explore cryptographic agent identity, formally verified policies, remote attestation, and autonomous cyber-defense under strict safety constraints.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

CHAPTER 08

Production architecture and frontier agent systems

Operate agentic systems as long-lived products with release discipline, capacity, cost, reliability, migration, and research boundaries.

Chapter operating position

Operate agentic systems as long-lived products with release discipline, capacity, cost, reliability, migration, and research boundaries.

8.1 Reference production architecture

Reference production architecture is treated here as an engineering capability rather than a vocabulary item. Combine API gateways, identity, policy, harnesses, model routers, tool fabrics, memory, durable state, telemetry, evidence, and human control. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for reference production architecture.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating reference production architecture as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for reference production architecture.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

8.2 Reliability, capacity, cost, and service objectives

Reliability, capacity, cost, and service objectives is treated here as an engineering capability rather than a vocabulary item. Define mission latency, completion, recovery, tool availability, model fallback, cost, and operator-effort objectives. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for reliability, capacity, cost, and service objectives.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Implementation sketch

```
mission:
  objective: "Implement the smallest verifiable change"
  context:
    include: [architecture, contracts, tests, affected_code]
    exclude: [secrets, unrelated_history]
  gates:
    - typecheck
    - unit_tests
    - security_review
    - evidence_bundle
```

Failure patterns

- Treating reliability, capacity, cost, and service objectives as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for reliability, capacity, cost, and service objectives.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

8.3 Release, compatibility, and change management

Release, compatibility, and change management is treated here as an engineering capability rather than a vocabulary item. Version prompts, tools, skills, models, schemas, memory, policies, and evaluators and canary changes against representative missions. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency, orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for release, compatibility, and change management.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating release, compatibility, and change management as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for release, compatibility, and change management.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

8.4 Frontier architectures and adoption gates

Frontier architectures and adoption gates is treated here as an engineering capability rather than a vocabulary item. Assess self-improving agents, world models, embodied agents, decentralized agents, and autonomous organizations with explicit readiness criteria. The design objective is to make the behavior understandable before it is automated, measurable before it is optimized, and recoverable before it is scaled.

Within MYTHOS-AI - Artificial Intelligence & Agentic Systems, this capability should be represented through explicit boundaries, typed or otherwise enforceable contracts, observable state transitions, and named owners. A production design separates intent from mechanism, control-plane decisions from data-plane execution, and nominal behavior from degraded or adversarial behavior. The resulting system is easier to test, review, migrate, and operate because its assumptions are visible rather than embedded in tribal knowledge.

Implementation begins with a small, inspectable baseline. Establish the authoritative inputs, expected outputs, invariants, timeout and retry behavior, identity context, telemetry, and rollback path. Only then add abstraction, concurrency,

orchestration, optimization, or autonomous behavior. This sequence prevents sophistication from hiding incomplete fundamentals and makes later evidence comparable across revisions.

Engineering practices

- Define the authoritative source and ownership boundary for frontier architectures and adoption gates.
- Make preconditions, invariants, error states, and recovery actions explicit.
- Instrument the critical path with stable identifiers, timestamps, and outcome codes.
- Validate in a representative environment before widening blast radius.
- Capture repeatable evidence so another engineer can reproduce the conclusion.

Failure patterns

- Treating frontier architectures and adoption gates as a product feature rather than an end-to-end engineering behavior.
- Automating an undocumented process and scaling its ambiguity.
- Relying on a happy-path demonstration without degraded-state or rollback testing.
- Using aggregate dashboards without retaining trace-level evidence.
- Allowing ownership, version, or authority to become implicit.

Validation and evidence

Method	Expected evidence
Examine	Architecture, configuration, contracts, runbooks, and source authority for frontier architectures and adoption gates.
Interview	Engineers and operators responsible for design, implementation, review, and incident ownership.
Test	Nominal, boundary, failure, timeout, recovery, and rollback scenarios with retained evidence.
Measure	Correctness, latency, reliability, resource use, security outcomes, and operator effort.

Chapter 08 laboratory and review

Practical laboratory

Design and capacity-test a production agent platform for concurrent coding, research, and infrastructure missions. Include degraded modes and model-provider failure.

Laboratory evidence package

- Architecture or topology showing boundaries, identities, dependencies, and authoritative inputs.
- Versioned configuration or code with explanatory comments and reproducible setup instructions.
- Nominal-path validation plus at least two injected failure or boundary conditions.
- Structured logs, traces, metrics, packet captures, test output, or other appropriate evidence.
- A concise decision record covering findings, limitations, residual risks, and next actions.

Frontier extension

Define an adoption dossier for a frontier agent technique, separating demonstrated capability, laboratory evidence, unresolved risk, and speculation.

Review gate

Advance only when another engineer can reproduce the result, explain the architecture, identify the failure boundary, and distinguish measured evidence from assumptions or projections.

Integrated learning roadmap

The guide is intended to be revisited. First establish fluency, then build a small implementation, operate it under failure, automate repeatable work, and finally evaluate emerging techniques against a stable baseline. Progress is demonstrated through evidence rather than reading completion alone.

Stage	Demonstrated capability
Foundation	Explain the system from first principles and trace its critical path without relying on product terminology.
Build	Implement the smallest representative system with explicit contracts, identities, telemetry, and tests.
Operate	Diagnose injected failures, recover safely, and preserve evidence of the causal chain.
Automate	Convert a proven manual workflow into bounded, idempotent, observable automation.
Architect	Design for scale, resilience, security, interoperability, and lifecycle change.
Explore	Evaluate frontier techniques using stated hypotheses, limitations, and adoption gates.

Source authority register

Source policy
Official standards, specifications, and project documentation remain with their issuing organizations. Mythos Systems provides original engineering interpretation, synthesis, implementation guidance, and relationship mapping. Source verification dates do not guarantee that an authority has not changed since review.

01. NIST - Artificial Intelligence Risk Management Framework 1.0
Role: AI governance, measurement, and risk-management foundation.
Verified: 2026-07-26
Official source: https://www.nist.gov/itl/ai-risk-management-framework

02. NIST - NIST AI 600-1 - Generative Artificial Intelligence Profile
Role: Generative-AI risk profile and control guidance.
Verified: 2026-07-26
Official source: https://doi.org/10.6028/NIST.AI.600-1

03. NIST - SP 800-53 Control Overlays for Securing AI Systems
Role: Security-control overlay work for AI systems.
Verified: 2026-07-26
Official source: https://csrc.nist.gov/projects/cosais

04. Model Context Protocol - Model Context Protocol Specification 2025-11-25
Role: Authoritative MCP lifecycle, capability, transport, resource, prompt, and tool protocol requirements.
Verified: 2026-07-26
Official source: https://modelcontextprotocol.io/specification/2025-11-25

05. OWASP - Top 10 for Agentic Applications 2026
Role: Agentic application threat taxonomy and mitigations.
Verified: 2026-07-26
Official source: https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

06. OWASP - Securing Agentic Applications Guide 1.0
Role: Practical secure design and deployment guidance for agentic applications.
Verified: 2026-07-26
Official source: https://genai.owasp.org/resource/securing-agentic-applications-guide-1-0/

07. OpenTelemetry - OpenTelemetry Semantic Conventions
Role: Vendor-neutral telemetry semantics for distributed, cloud-native, and AI systems.
Verified: 2026-07-26
Official source: https://opentelemetry.io/docs/specs/semconv/

08. SLSA Project - Supply-chain Levels for Software Artifacts Specification
Role: Software supply-chain integrity, provenance, build, source, and verification requirements.
Verified: 2026-07-26
Official source: https://slsa.dev/spec/

Authority application and refresh triggers

Authority class	Required library action
Protocols and specifications	Recheck interoperability, security considerations, defaults, and migration guidance when a referenced specification changes.
Security and assurance frameworks	Update control mappings, evidence expectations, risk language, and assessment procedures after material framework revision.
Languages, runtimes, and projects	Re-run examples, tests, deprecation checks, and operational recommendations against supported releases.
Benchmarks and evaluations	Re-execute claims when workloads, methods, models, compilers, drivers, hardware, or scoring rules change.
Operational evidence	Incorporate validated incidents, failure tests, reader corrections, and measured implementation outcomes into the next revision.

Limitations, freshness, and revision control

Boundary	Statement
Publication limitation	This guide synthesizes broad engineering practice. It does not replace product documentation, legal advice, safety certification, or environment-specific design review.
Evidence limitation	Not every pattern is benchmarked in every environment. Examples illustrate engineering methods and must be validated against actual versions, workloads, hardware, policy, and failure conditions.
Freshness limitation	Vendor behavior, software libraries, AI models, security threats, and emerging standards can change quickly. Volatile claims require source revalidation before operational adoption.
Adoption limitation	Frontier content is not an implicit adoption recommendation. Adoption requires defined value, qualification gates, controlled proof, rollback, ownership, and residual-risk acceptance.
Status boundary	Candidate Living Guide describes the publication, not certification of a product, implementation, engineer, or organization.

Revision record

Version	Revision statement
1.0.0-candidate	Initial candidate living-guide edition. Published 2026-07-26; scheduled review 2026-10-26.

Search and relationship metadata

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-SWE; MYTHOS-DAT; MYTHOS-SEC; MYTHOS-CLD
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	agentic systems, agent harnesses, MCP, tools, skills, memory, context engineering, multi-agent, multi-model, parallelism, governance, evaluation, observability
Canonical route	/library/field-guides/mythos-guide-agent-0001/

Living publication

Corrections, expanded laboratories, improved diagrams, authority changes, and new engineering evidence should revise this permanent publication rather than create disconnected replacements.