



CANONICAL LIVING OVERVIEW GUIDE

AI-Driven Software Development and Agentic Engineering

A governed engineering guide to model selection, specification-driven development, repository context, coding agents, multi-agent workflows, testing, review, security, provenance, hallucination control, and verified completion.

PERMANENT PUBLICATION IDENTITY

AI-Driven Software Development and Agentic Engineering

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

Reader orientation

Dimension	Engineering position
Primary domain	MYTHOS-AI - Artificial Intelligence & Agentic Systems
Secondary domain	MYTHOS-SWE - Software Engineering & Design
Audience	Software engineers, AI-assisted developers, architects, technical leads, agent-platform builders, reviewers, security engineers, and engineering managers.
Prerequisites	Working knowledge of software engineering and source control. Familiarity with language models, prompts, APIs, testing, and CI/CD is helpful.
Publisher / owner	Mythos Systems / Aaron Stovall
Public classification	Public technical guidance

Expected outcomes

- Select models and agent configurations according to task, evidence, risk, cost, context, and verification needs.
- Convert product intent into specifications, plans, contracts, tasks, tests, and acceptance evidence suitable for human and agent execution.
- Construct repository context and memory without flooding models or losing authoritative constraints.
- Operate single- and multi-agent development workflows with bounded authority, checkpoints, independent review, and rollback.
- Distinguish generated output from verified completion through tests, provenance, review, and evidence bundles.

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

How to use this guide	3
Contents	3
Chapter 01 - AI-assisted engineering foundations	5
Chapter 01 laboratory and review	9
Chapter 02 - Model selection, routing, and operational economics	10
Chapter 02 laboratory and review	14
Chapter 03 - Specification-driven development and planning	15
Chapter 03 laboratory and review	19
Chapter 04 - Repository context, memory, and knowledge systems	20
Chapter 04 laboratory and review	24
Chapter 05 - Coding agents, tools, and execution harnesses	25
Chapter 05 laboratory and review	29

Chapter 06 - Multi-agent and parallel development	30
Chapter 06 laboratory and review	34
Chapter 07 - Testing, review, security, and hallucination control	35
Chapter 07 laboratory and review	39
Chapter 08 - Delivery, governance, and verified outcomes	40
Chapter 08 laboratory and review	44
Integrated learning roadmap	45
Source authority register	46
Limitations, freshness, and revision control	47

CHAPTER 01

AI-assisted engineering foundations

Treat AI development systems as probabilistic collaborators operating inside deterministic engineering and governance boundaries.

Chapter operating position

Treat AI development systems as probabilistic collaborators operating inside deterministic engineering and governance boundaries.

1.1 Capabilities, limitations, and task fit

Capabilities, limitations, and task fit is treated here as an engineering capability rather than a vocabulary item. Match model and tool capabilities to planning, coding, debugging, documentation, testing, review, or research without assuming general intelligence. 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 capabilities, limitations, and task fit.

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 capabilities, limitations, and task fit 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 capabilities, limitations, and task fit.
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 Probabilistic generation and deterministic gates

Probabilistic generation and deterministic gates is treated here as an engineering capability rather than a vocabulary item. Use models to propose and transform while compilers, schemas, tests, policies, reviewers, and runtime evidence determine acceptance. 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 probabilistic generation and deterministic 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.

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 probabilistic generation and deterministic 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 probabilistic generation and deterministic 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.

1.3 Human authority and responsibility

Human authority and responsibility is treated here as an engineering capability rather than a vocabulary item. Keep ownership, approval, risk acceptance, production access, and professional judgment attributable to people or explicitly governed roles. 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 human authority and responsibility.

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 human authority and responsibility 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 human authority and responsibility.
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 Completion, confidence, and evidence

Completion, confidence, and evidence is treated here as an engineering capability rather than a vocabulary item. Define done through acceptance criteria and retained evidence rather than fluent explanations or agent self-report. 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 completion, confidence, and evidence.
- 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 completion, confidence, and evidence 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 completion, confidence, and evidence.
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

Take one previous AI-assisted coding task and reconstruct the difference between generated output, reviewed output, tested output, deployed output, and verified outcome.

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 proof-carrying agent work, formalized completion criteria, calibrated confidence, and development systems that must cite every repository and runtime claim.

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

Model selection, routing, and operational economics

Choose and route models using task-specific capability, latency, cost, privacy, context, reliability, and evidence requirements.

Chapter operating position

Choose and route models using task-specific capability, latency, cost, privacy, context, reliability, and evidence requirements.

2.1 Task taxonomy and capability profiles

Task taxonomy and capability profiles is treated here as an engineering capability rather than a vocabulary item. Classify planning, architecture, code generation, repair, review, testing, documentation, search, and operational tasks separately. 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 taxonomy and capability profiles.

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 taxonomy and capability profiles 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 taxonomy and capability profiles.
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 Local, hosted, and hybrid model strategy

Local, hosted, and hybrid model strategy is treated here as an engineering capability rather than a vocabulary item. Balance sovereignty, speed, cost, context, tool integration, quality, and sensitive-data constraints across deployment modes. 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 local, hosted, and hybrid model strategy.

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 local, hosted, and hybrid model strategy 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 local, hosted, and hybrid model strategy.
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 Model routing and escalation

Model routing and escalation is treated here as an engineering capability rather than a vocabulary item. Start with the lowest sufficient capability, detect uncertainty or failure, escalate deliberately, and preserve task history 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 model routing and escalation.
- 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 model routing and escalation 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 model routing and escalation.
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 Budgets, quotas, latency, and energy

Budgets, quotas, latency, and energy is treated here as an engineering capability rather than a vocabulary item. Make token, time, concurrency, monetary, compute, and energy budgets visible and tied to engineering value. 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 budgets, quotas, latency, and energy.
- 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 budgets, quotas, latency, and energy 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 budgets, quotas, latency, and energy.
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

Create a model-routing matrix for ten development tasks. Run representative tasks through at least two model classes and compare quality, latency, cost, privacy, and verification effort.

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 mixture-of-model systems, speculative collaboration, confidence-aware routing, local frontier models, and hardware-aware orchestration.

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

Specification-driven development and planning

Transform ambiguous requests into stable intent, architecture, contracts, milestones, tests, and risk controls before autonomous implementation.

Chapter operating position

Transform ambiguous requests into stable intent, architecture, contracts, milestones, tests, and risk controls before autonomous implementation.

3.1 Product intent and acceptance criteria

Product intent and acceptance criteria is treated here as an engineering capability rather than a vocabulary item. Define users, outcomes, constraints, non-goals, quality attributes, safety, security, and observable completion. 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 product intent and acceptance criteria.

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 product intent and acceptance criteria 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 product intent and acceptance criteria.
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 Architecture and contract packets

Architecture and contract packets is treated here as an engineering capability rather than a vocabulary item. Provide system context, boundaries, data models, APIs, invariants, dependencies, and decisions that agents can reference without inventing them. 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 architecture and contract packets.

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 architecture and contract packets 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 architecture and contract packets.
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 Task decomposition and dependency graphs

Task decomposition and dependency graphs is treated here as an engineering capability rather than a vocabulary item. Break work into bounded tasks with inputs, outputs, ownership, parallelism, prerequisites, gates, and integration points. 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 decomposition and dependency 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 task decomposition and dependency 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 task decomposition and dependency 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.

3.4 Plan review and adversarial critique

Plan review and adversarial critique is treated here as an engineering capability rather than a vocabulary item. Use independent review to expose missing requirements, unsafe assumptions, hidden coupling, impossible sequencing, and weak tests. 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 review and adversarial critique.
- 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 review and adversarial critique 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 review and adversarial critique.
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

Write an agent-ready implementation packet for a small application feature, including requirements, architecture, interfaces, task graph, test plan, risk register, and completion 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 executable specifications, model-checked plans, typed mission languages, formal workflow compilation, and adaptive planning constrained by immutable goals.

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

Repository context, memory, and knowledge systems

Provide the minimum authoritative context needed for correct work while preserving lineage, freshness, and retrieval quality.

Chapter operating position

Provide the minimum authoritative context needed for correct work while preserving lineage, freshness, and retrieval quality.

4.1 Repository maps and code intelligence

Repository maps and code intelligence is treated here as an engineering capability rather than a vocabulary item. Use file structure, symbols, references, ownership, tests, build graph, runtime boundaries, and recent changes to orient the agent. 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 repository maps and code intelligence.

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 repository maps and code intelligence 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 repository maps and code intelligence.
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 Context selection and token discipline

Context selection and token discipline is treated here as an engineering capability rather than a vocabulary item. Prioritize task-relevant contracts, affected code, examples, tests, decisions, and errors instead of indiscriminate repository dumps. 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 selection and token discipline.

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 context selection and token discipline 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 selection and token discipline.
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 Summaries, memory, and freshness

Summaries, memory, and freshness is treated here as an engineering capability rather than a vocabulary item. Separate raw source, derived summaries, session memory, durable project memory, and user preferences with version and confidence. 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 summaries, memory, 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 summaries, memory, 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 summaries, memory, 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 Retrieval, grounding, and source authority

Retrieval, grounding, and source authority is treated here as an engineering capability rather than a vocabulary item. Require citations to repository locations, documentation, issues, runtime evidence, and external primary sources for material claims. 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, and source 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 retrieval, grounding, and source 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 retrieval, grounding, and source 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 04 laboratory and review

Practical laboratory

Build a context packet for a nontrivial repository task. Compare a naive full-context approach with a curated map, retrieved slices, decisions, tests, and error 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 code knowledge graphs, temporal repository memory, semantic caches, retrieval evaluation, privacy-preserving embeddings, and self-correcting context selection.

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

Coding agents, tools, and execution harnesses

Give agents useful capabilities through typed tools, sandboxes, permissions, timeouts, checkpoints, and observable execution.

Chapter operating position

Give agents useful capabilities through typed tools, sandboxes, permissions, timeouts, checkpoints, and observable execution.

5.1 Tool contracts and capability boundaries

Tool contracts and capability boundaries is treated here as an engineering capability rather than a vocabulary item. Define inputs, outputs, side effects, permissions, validation, idempotency, timeout, resource limits, and evidence for every tool. 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 capability boundaries.

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 capability boundaries 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 capability boundaries.
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 Sandboxing and workspace isolation

Sandboxing and workspace isolation is treated here as an engineering capability rather than a vocabulary item. Separate tasks, branches, filesystems, processes, credentials, networks, and caches to contain mistakes and support reproducibility. 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 sandboxing and workspace isolation.

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 sandboxing and workspace isolation 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 sandboxing and workspace isolation.
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 Checkpoints, commits, and rollback

Checkpoints, commits, and rollback is treated here as an engineering capability rather than a vocabulary item. Create reviewable increments with tests, diffs, rationale, and restore points rather than one opaque burst of changes. 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, commits, 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, commits, 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, commits, 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.

5.4 Execution telemetry and provenance

Execution telemetry and provenance is treated here as an engineering capability rather than a vocabulary item. Record model, prompt, context, tool calls, files, commands, outputs, tests, decisions, cost, and final artifacts. 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 execution telemetry and provenance.
- 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 execution telemetry and provenance 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 execution telemetry and provenance.
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

Configure a coding-agent workflow in an isolated workspace with scoped tools, branch discipline, checkpoints, test gates, command logging, and a recoverable failed attempt.

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 WebAssembly tool sandboxes, capability security, cryptographically signed agent actions, reproducible workspaces, and policy-enforced tool marketplaces.

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

Multi-agent and parallel development

Use specialization and parallelism only where tasks can be isolated, independently verified, and integrated through explicit contracts.

Chapter operating position

Use specialization and parallelism only where tasks can be isolated, independently verified, and integrated through explicit contracts.

6.1 Supervisors, specialists, and role design

Supervisors, specialists, and role design is treated here as an engineering capability rather than a vocabulary item. Assign planning, architecture, implementation, testing, security, documentation, and review roles with distinct authority 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 supervisors, specialists, and role design.

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 supervisors, specialists, and role design 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 supervisors, specialists, and role design.
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 Fan-out, fan-in, and dependency-aware parallelism

Fan-out, fan-in, and dependency-aware parallelism is treated here as an engineering capability rather than a vocabulary item. Parallelize research, isolated modules, tests, reviews, or alternatives while controlling shared files and integration order. 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, and dependency-aware parallelism.

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 fan-out, fan-in, and dependency-aware parallelism 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, and dependency-aware parallelism.
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 Independent verification and debate

Independent verification and debate is treated here as an engineering capability rather than a vocabulary item. Use separate agents or models to challenge plans, inspect diffs, run tests, analyze security, and compare conclusions against 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 independent verification and debate.
- 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 independent verification and debate 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 independent verification and debate.
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 Conflict, merge, and consensus control

Conflict, merge, and consensus control is treated here as an engineering capability rather than a vocabulary item. Resolve incompatible changes through contracts, ownership, branch integration, tests, and human decision rather than model voting alone. 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 conflict, merge, and consensus control.
- 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 conflict, merge, and consensus control 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 conflict, merge, and consensus control.
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

Run a multi-agent feature build with planner, two implementers, tester, security reviewer, and integrator roles. Preserve prompts, outputs, conflicts, decisions, tests, and final 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 large agent arrays, dynamic teams, blackboard systems, market-style task allocation, heterogeneous models, and formally constrained coordination.

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

Testing, review, security, and hallucination control

Assume generated work may be plausible but wrong and require independent evidence at every consequential boundary.

Chapter operating position

Assume generated work may be plausible but wrong and require independent evidence at every consequential boundary.



7.1 Compiler, type, lint, and static-analysis gates

Compiler, type, lint, and static-analysis gates is treated here as an engineering capability rather than a vocabulary item. Use language and repository tooling as fast deterministic checks and treat failures as evidence for iterative repair. 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 compiler, type, lint, and static-analysis 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 compiler, type, lint, and static-analysis 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 compiler, type, lint, and static-analysis 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.

7.2 Test generation and test adequacy

Test generation and test adequacy is treated here as an engineering capability rather than a vocabulary item. Generate tests from requirements and risk, then evaluate them with coverage, mutation, adversarial cases, and independent review. 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 test generation and test adequacy.

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 test generation and test adequacy 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 test generation and test adequacy.
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 Security review and dependency control

Security review and dependency control is treated here as an engineering capability rather than a vocabulary item. Inspect authorization, secrets, injection, unsafe execution, dependencies, licenses, provenance, and tool permissions. 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 security review and dependency control.

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 security review and dependency control 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 security review and dependency control.
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 Hallucination and unsupported-claim containment

Hallucination and unsupported-claim containment is treated here as an engineering capability rather than a vocabulary item. Require file, symbol, command, test, source, or runtime citations and reject invented APIs, success reports, and environmental assumptions. 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 hallucination and unsupported-claim 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 hallucination and unsupported-claim 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 hallucination and unsupported-claim 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 07 laboratory and review

Practical laboratory

Provide an agent with a defect and incomplete specification. Measure invented assumptions, require evidence citations, add deterministic gates, and compare the ungoverned and governed outcomes.

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 adversarial agent evaluation, generated-code proof obligations, runtime attestation, calibrated uncertainty, and models trained on evidence-backed engineering trajectories.

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

Delivery, governance, and verified outcomes

Integrate agent work into normal software delivery with ownership, policy, auditability, operational validation, and lifecycle learning.

Chapter operating position

Integrate agent work into normal software delivery with ownership, policy, auditability, operational validation, and lifecycle learning.

8.1 Pull requests and review artifacts

Pull requests and review artifacts is treated here as an engineering capability rather than a vocabulary item. Require focused diffs, rationale, tests, risk, migration, rollback, and generated-content disclosure appropriate to the 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 pull requests and review artifacts.

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 pull requests and review artifacts 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 pull requests and review artifacts.
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 CI/CD and production boundaries

CI/CD and production boundaries is treated here as an engineering capability rather than a vocabulary item. Keep deployment credentials, release authority, policy exceptions, and high-risk actions behind explicit gates 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 ci/cd and production boundaries.

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 ci/cd and production boundaries 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 ci/cd and production boundaries.
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 Evidence bundles and completion records

Evidence bundles and completion records is treated here as an engineering capability rather than a vocabulary item. Package requirements, plan, changes, tests, scans, reviews, deployment, observations, limitations, and approvals as the outcome record. 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 completion records.

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 completion records 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 completion records.
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 Metrics, feedback, and model improvement

Metrics, feedback, and model improvement is treated here as an engineering capability rather than a vocabulary item. Measure escaped defects, review effort, cycle time, test quality, rework, security, cost, and user outcomes rather than generated code volume. 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 metrics, feedback, and model improvement.
- 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 metrics, feedback, and model improvement 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 metrics, feedback, and model improvement.
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

Deliver a complete AI-assisted change through pull request, deterministic CI, human review, canary, production validation, rollback readiness, and an evidence bundle that another engineer can audit.

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 autonomous delivery under policy, model reputation systems, reusable verified trajectories, continuous agent evaluation, and organizational governance for increasingly capable systems.

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 - SP 800-218 - Secure Software Development Framework Role: Secure software development practices. Verified: 2026-07-26 Official source: https://csrc.nist.gov/pubs/sp/800/218/final
02. NIST - Artificial Intelligence Risk Management Framework 1.0 Role: AI risk, governance, measurement, and management framework. Verified: 2026-07-26 Official source: https://www.nist.gov/itl/ai-risk-management-framework
03. NIST - AI 600-1 - Generative Artificial Intelligence Profile Role: Generative-AI risk profile and action guidance. Verified: 2026-07-26 Official source: https://doi.org/10.6028/NIST.AI.600-1
04. OWASP - Top 10 for Large Language Model Applications 2025 Role: LLM and generative-AI application security risks. Verified: 2026-07-26 Official source: https://genai.owasp.org/llm-top-10/
05. Model Context Protocol - Model Context Protocol Specification Role: Tool and context interoperability protocol reference. Verified: 2026-07-26 Official source: https://modelcontextprotocol.io/specification/2025-11-25
06. OpenTelemetry - Semantic Conventions Role: Telemetry semantic conventions for distributed systems and generative AI. Verified: 2026-07-26 Official source: https://opentelemetry.io/docs/specs/semconv/
07. MLCommons - MLPerf Training and Inference Benchmarks Role: Reproducible model performance and system benchmarking reference. Verified: 2026-07-26 Official source: https://mlcommons.org/benchmarks/
08. OWASP - Application Security Verification Standard Role: Application-security verification requirements. Verified: 2026-07-26 Official source: https://owasp.org/www-project-application-security-verification-standard/

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 - Software Engineering & Design
Publication class	Field Guide / Canonical Living Overview Guide
Skill levels	L1 Foundational; L2 Practitioner; L3 Advanced; L4 Frontier
Tags	AI-driven development, agentic engineering, coding agents, model routing, context engineering, multi-agent, testing, provenance, hallucination control, verified completion
Canonical route	/library/field-guides/mythos-guide-aidev-0001/

Living publication

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