ETIS WHITE PAPER SERIES

WP-008

Operational Readiness

Engineering Systems That Can Be Observed, Recovered, Governed, and Sustained in Production

Core Thesis

A system is not operationally ready because it passed testing or deployed successfully. It is ready only when the organization can understand its behavior, detect failure, intervene safely, recover service, govern change, and sustain accountable ownership under real operating conditions.

William T. O'Connell, Ph.D.

Adjunct Professor of Computer Science, Loyola University Chicago

Former IBM Distinguished Engineer and CTO, IBM Consulting Quality and Delivery

Creator, Engineering Trustworthy Intelligent Systems (ETIS)

July 2026

ETIS WP-008 | Version 1.0

Executive Abstract

Software organizations routinely treat deployment as the finish line. A build passes automated checks, a release is approved, and the system reaches production. Yet many of the most damaging failures occur after this point, when real users, real data, unpredictable dependencies, operational load, and human response expose weaknesses that preproduction validation could not fully reveal. The decisive question is therefore not whether a system can be deployed. It is whether the organization is prepared to operate it responsibly.

Operational readiness is the engineered condition in which a system, its operating team, and its governance environment are prepared for production consequence. It includes observable behavior, service objectives, ownership, runbooks, alerting, deployment and rollback controls, incident response, recovery, capacity, dependency management, security operations, data protection, and post-release learning. For intelligent and agentic systems, readiness also includes monitoring model and agent behavior, tool use, policy compliance, cost, context, memory, human escalation, and the ability to contain or deactivate unsafe execution.

This paper synthesizes established operational practices from Google Site Reliability Engineering, the AWS and Azure Well-Architected frameworks, DORA research, OpenTelemetry, CISA incident-response guidance, NIST AI risk-management guidance, and emerging agentic-security work. It then develops the ETIS position: operational readiness is not a late checklist. It is a lifecycle property designed into requirements and architecture, supported by evidence at release, tested through operational exercises, and continuously revalidated through production telemetry and learning.

Executive Takeaway
Release approval permits change. Operational readiness demonstrates that the organization can live with the consequences of that change.

1. Production Is a Different Engineering Environment

Preproduction environments are controlled approximations. Production is where incomplete knowledge becomes consequence. Real users behave differently from test personas. Data is larger, messier, and historically inconsistent. Dependencies slow down or fail. Traffic patterns shift. Operators make decisions under time pressure. Security threats adapt. Small configuration differences become large operational effects. A system that performed correctly in a test environment may still be unready for the environment in which business, safety, reputation, and public trust are actually at stake.

Google Site Reliability Engineering treats production readiness as a prerequisite for taking responsibility for a service. Its Production Readiness Review model examines the reliability needs of a service before an SRE team accepts operational responsibility.[1] AWS and Azure similarly place operational excellence alongside reliability, security, performance, and cost as a fundamental quality concern, not an afterthought.[2][3] The shared message is direct: the ability to operate a system is part of the system design.

Operational readiness therefore differs from deployment success. Deployment answers whether a change reached an environment. Readiness answers whether the system can be understood, controlled, supported, and recovered once it is there. A deployment can succeed while users experience failure. A service can remain technically available while producing wrong decisions. An AI agent can respond quickly while violating authority boundaries. Operational engineering must evaluate outcomes, not only process completion.

Operational Principle
Production is not merely another environment. It is the point at which engineering claims encounter reality.

2. Operational Readiness Begins Before Release

The common mistake is to begin readiness work when release is near. At that point, many operational properties are already constrained by earlier decisions. If requirements do not define availability, recovery, auditability, human intervention, or acceptable degradation, teams cannot test them meaningfully. If architecture does not expose health, isolate failure, preserve state, or support rollback, operators cannot manufacture those capabilities through a runbook.

Operational requirements are business requirements. Azure's Operational Excellence guidance explicitly states that workload operational requirements are as important as business requirements.[3] In ETIS, operational readiness begins when teams identify who will own the system, what outcomes matter, how failure will be detected, which actions are reversible, what data must be protected, and what evidence will be required before release. Architecture then converts those expectations into boundaries, telemetry, recovery paths, failure containment, and governance controls.

This lifecycle view changes the role of release readiness. The release gate is not where operational concerns are introduced. It is where accumulated evidence is judged: monitoring is implemented, alerts are actionable, recovery has been exercised, operators understand the system, dependencies are known, and residual risk has an accountable owner. Readiness cannot be documented into existence at the end; it must be engineered throughout the lifecycle.

3. The Dimensions of Readiness

Operational readiness is multidimensional. A system can be strong in one dimension and dangerously weak in another. For example, it may have automated deployment but no credible rollback, extensive logs but no user-centered service objectives, or a documented incident process without anyone trained to execute it. A readiness judgment must consider the whole operating system around the software.

Readiness Dimension	Core Question
Service and user outcomes	What must remain true for users and the business, and how will degradation be recognized?
Observability	Can operators explain current behavior and investigate unfamiliar failure?
Ownership and support	Who is accountable, who responds, and do they have the authority and competence to act?
Release and recovery	Can change be introduced gradually, stopped, rolled back, or repaired safely?
Reliability and capacity	Can the system tolerate expected load, dependency failure, and resource pressure?
Security and data operations	Can threats, misuse, leakage, and policy violations be detected and contained?
Incident response	Can the organization coordinate diagnosis, communication, mitigation, and recovery?
Governance and evidence	Are decisions, exceptions, residual risks, and operational outcomes visible and reviewable?

Readiness Dimension	Core Question
Sustainment	Can the system be maintained, improved, transferred, and eventually retired responsibly?

The dimensions reinforce one another. Observability without ownership produces dashboards that no one acts upon. Alerting without service objectives produces noise. Automation without rollback accelerates failure. Incident response without architectural understanding prolongs recovery. Readiness is the alignment of technical capability, human capability, and governance.

4. Observability as Operational Evidence

Monitoring asks whether known conditions have occurred. Observability is broader: it allows engineers to ask new questions about system behavior, including questions they did not know in advance. OpenTelemetry defines observability as the ability to understand a system's internal state through outputs such as traces, metrics, and logs, and emphasizes instrumentation as the means of producing that evidence.[4][5]

Telemetry becomes operational evidence when it supports a decision. A metric matters because it indicates user impact, capacity risk, policy violation, or abnormal behavior. A trace matters because it connects a failed outcome across services and dependencies. A log matters because it preserves enough context to explain an event without exposing protected data. The goal is not maximum data collection. It is useful visibility with known ownership, retention, privacy, and cost.

Operationally ready systems connect telemetry to the claims made about them. If the release claims a workflow is reliable, telemetry should show completion, failure, latency, and retry behavior. If the architecture claims human approval is required for a high-risk action, operations should expose approval and bypass events. If an AI system claims bounded tool access, runtime evidence should reveal which tool was called, by which identity, with what result, and under which policy.

Evidence Principle
A dashboard is not operational proof. Operational proof exists when telemetry supports diagnosis, intervention, governance, and learning.

5. Service Objectives, User Impact, and Decision Thresholds

Operational readiness requires a definition of acceptable service. Without one, teams cannot distinguish normal variation from failure, prioritize incidents, or decide when to stop a release. Site Reliability Engineering uses service-level indicators and objectives to connect system behavior to user expectations. DORA likewise distinguishes delivery throughput from instability and treats recovery from failed deployment as a core performance outcome.[6]

A service objective should represent what users or dependent systems actually experience: successful completion, latency, correctness, data freshness, availability, or another meaningful outcome. Internal resource metrics may explain a problem, but they are not substitutes for the outcome. Intelligent systems may require additional objectives for unsafe-action rate, escalation behavior, unsupported-output rate, policy compliance, cost, or the quality of decisions over time.

Thresholds turn observation into action. Teams should know which conditions trigger investigation, rollback, traffic reduction, model or feature disablement, human escalation, or incident declaration. These thresholds should be defined before the system is under pressure. The purpose is not to automate every decision, but to reduce ambiguity when time and attention are scarce.

6. Release, Rollback, and Controlled Exposure

A trustworthy release process assumes that change can be wrong. That assumption leads to smaller changes, automated verification, staged exposure, canarying, feature controls, backward compatibility, and tested recovery. Google SRE includes canarying and launch coordination among its operational practices, while cloud well-architected guidance emphasizes safe, repeatable, and observable change.[2][7]

Rollback is often discussed as a technical mechanism, but it is an organizational capability. Teams must know what can be reversed, which data changes are irreversible, who may authorize rollback, how long recovery takes, and what happens when the old version is no longer compatible with changed state. In many systems, roll-forward repair is more realistic than rollback. Readiness requires an explicit strategy rather than a generic statement that the team "can roll back."

Controlled exposure is especially important for AI-enabled behavior. A prompt, policy, model, retrieval source, routing rule, tool permission, or memory configuration may change behavior without a traditional code deployment. These elements need versioning, validation, release records, and disablement paths. The operational release unit is the full behavior-producing configuration, not only the source code.

7. Incident Response, Recovery, and Organizational Memory

Every system eventually encounters conditions its designers did not fully anticipate. Operational readiness accepts this reality and prepares the organization to respond. CISA incident-response playbooks emphasize standardized phases and clear responsibilities for detection, coordination, containment, recovery, and completion.[8] Google SRE similarly treats on-call, incident response, and blameless postmortems as core operational capabilities.[9]

An incident process must be executable under stress. Roles, communication channels, decision authority, escalation, status reporting, evidence preservation, and handoff should be understood before an incident begins. Runbooks should assist diagnosis and safe action, but they cannot replace engineering understanding. Operators need enough architectural context to recognize when a documented procedure no longer fits the failure.

Recovery is not complete when service returns. The organization must understand what happened, what evidence supports that understanding, what residual risk remains, and what changes are required. A postmortem converts operational experience into organizational memory. When findings update requirements, architecture, tests, controls, runbooks, and future reviews, the system becomes more trustworthy. When the report is filed and forgotten, the organization merely records failure.

8. Operational Readiness for AI and Agentic Systems

AI and agentic systems expand the operational surface. Teams must monitor not only infrastructure and application behavior, but also model outputs, context quality, retrieval behavior, tool calls, permissions, memory, routing, cost, policy compliance, and human escalation. The NIST AI Risk Management Framework emphasizes ongoing measurement and management across the AI lifecycle, including monitoring for performance degradation, unusual behavior, impacts, and the need to deactivate systems that operate outside intended limits.[10][11]

Agentic systems require special attention because they can take action. Runtime readiness includes identity for agents and tools, least privilege, action validation, containment, rate limits, audit trails, intervention, and termination capability. Emerging OWASP agentic-security work highlights risks such as goal hijacking, tool misuse, identity and privilege abuse, memory poisoning, insecure inter-agent communication, cascading failure, and rogue behavior.[12]

Human oversight must be operational, not symbolic. The human must receive sufficient context, have the authority to intervene, and be able to stop or reverse the action within a meaningful timeframe. A system that

asks for approval after the consequential action has already occurred is not meaningfully supervised. Readiness therefore depends on the relationship between system speed, action reversibility, detection latency, and human response capability.

Operational Signal	Why It Matters
Model and agent version	Connects observed behavior to the released configuration
Context and retrieval source	Reveals what information grounded the action
Tool call and authority	Shows what was attempted, permitted, denied, or changed
Human approval and override	Demonstrates meaningful oversight and intervention
Outcome quality and exception rate	Tests whether behavior remains within intended limits
Cost and resource consumption	Detects runaway execution and operational inefficiency
Policy and safety events	Exposes governance violations and containment actions

9. Ownership, Runbooks, and Human Capability

Systems do not operate themselves, even when automation is extensive. Someone must own service health, risk decisions, incident coordination, stakeholder communication, and improvement. Ownership must be explicit enough that another engineer can determine who is accountable without relying on private organizational knowledge.

Runbooks are valuable when they preserve tested procedures, prerequisites, expected outcomes, escalation, and recovery. They are dangerous when they create false confidence. A runbook that has never been exercised may fail when dependencies, credentials, interfaces, or organizational roles have changed. Operational readiness should include walkthroughs, simulations, or game days that test whether people can use the documented process.

Google SRE uses readiness for on-call as a practical proxy for system understanding: a person prepared to respond must know enough about the system to diagnose and act responsibly.[13] The same principle applies beyond formal SRE organizations. A team is not ready to operate a service if no one understands its architecture, data, dependencies, deployment, failure modes, and recovery constraints.

10. Resilience, Capacity, and Dependency Failure

Operational readiness must account for the system around the system. Cloud services, identity providers, databases, models, external APIs, data pipelines, and human processes all introduce dependencies. A design may be correct under normal conditions and fail catastrophically when a dependency slows, returns partial results, changes behavior, or becomes unavailable.

Resilience begins with explicit failure assumptions. Teams should identify critical dependencies, timeout and retry behavior, fallback paths, data-consistency consequences, capacity limits, single points of failure, and the conditions under which degraded operation is safer than continued execution. Reliability guidance from AWS and Azure emphasizes designing for failure, automation, repeatability, observability, and continuous improvement.[2][3]

Capacity readiness is not limited to peak traffic. It includes queue growth, storage, token and model quotas, third-party rate limits, human approval bottlenecks, and the cost of abnormal behavior. Agentic workflows can amplify load by generating recursive or parallel activity. The system must constrain fan-out, retries, and autonomous loops before resource pressure becomes business or safety impact.

11. Operational Governance and Continuous Assurance

Operational governance defines who may change the system, which risks require escalation, what evidence must be retained, and when service should be constrained or stopped. It is not separate from operations. It is embedded in release controls, access, alerting, policy enforcement, exception handling, incident authority, and review.

Readiness also expires. A service that was ready at launch may become unready as traffic, dependencies, personnel, regulation, data, models, or business purpose change. Continuous assurance compares current behavior and operating capability with the assumptions under which approval was granted. NIST's AI RMF and ISO/IEC 42001 both emphasize lifecycle monitoring, feedback, corrective action, and continual improvement rather than one-time approval.[10][14]

The repository should preserve the operational evidence chain: readiness review, release baseline, runbook, service objectives, dashboard and alert references, dependency inventory, incidents, postmortems, residual risks, exceptions, and improvement work. This record makes operations reviewable and gives future engineers the context needed to understand why controls exist.

12. Failure Modes and Readiness Theater

Readiness theater occurs when an organization produces the appearance of operational discipline without building the capability. A checklist is completed, a dashboard exists, a runbook is uploaded, or an owner is named, but the system remains difficult to understand and unsafe to operate.

Failure Mode	What It Looks Like	Operational Consequence
Deployment equals readiness	A successful pipeline is treated as proof of operational fitness	User-impacting weaknesses appear after release
Telemetry without decisions	Many dashboards and alerts, but no thresholds or owners	Noise obscures meaningful degradation
Untested recovery	Rollback, backup, or failover exists only on paper	Recovery fails when it is needed
Runbook as substitute for knowledge	Operators follow stale procedures without understanding architecture	Incidents escalate through incorrect action
Unknown dependencies	External services and data paths are incompletely mapped	Failures propagate unexpectedly
Human-in-the-loop theater	Humans approve without context, authority, or time	Unsafe autonomy appears governed
Unowned residual risk	Known limitations have no accountable owner or review date	Temporary acceptance becomes permanent exposure
Postmortem without change	Incidents are documented but findings do not alter engineering	The same failure returns

The remedy is not more documentation. It is tested capability. Readiness evidence should show that teams can observe, decide, intervene, recover, and learn. The stronger the system's authority and impact, the stronger that proof should be.

13. An Operational Readiness Operating Model

A practical operating model connects lifecycle work to an explicit readiness decision. Teams define operational requirements early, implement and verify operational capabilities, assemble a readiness record, conduct a risk-proportionate review, and then revalidate readiness through production evidence. The model should be lightweight for low-risk systems and progressively stronger for systems with broader impact, sensitive data, high availability needs, or autonomous authority.

Operating-Model Element	Design Question
Readiness owner	Who assembles the evidence and is accountable for unresolved gaps?
Operational requirements	What service, recovery, security, governance, and support outcomes must be met?
Evidence contract	What proof must exist before release: telemetry, tests, drills, runbooks, ownership, risk?
Review and decision rights	Who may approve, defer, constrain, or stop the release?
Controlled launch	How will exposure be staged, monitored, and reversed?
Operational handoff	Who accepts ongoing responsibility, and what knowledge and access are transferred?
Continuous reassessment	Which production signals or changes trigger renewed review?

Metrics should measure outcomes and operating capability rather than the existence of artifacts. Useful measures may include service-objective attainment, alert actionability, failed-deployment recovery time, change failure rate, incident recurrence, time to detect, time to contain, recovery exercise success, finding closure, dependency-related incidents, and the proportion of high-risk AI actions receiving effective human intervention. DORA's current metrics intentionally balance throughput and instability, reinforcing that fast delivery without controlled recovery is not high performance.[6]

14. A Five-Level Operational Readiness Maturity Model

Maturity Level	Characteristics
Level 1 — Reactive	Operations begin after deployment; ownership, telemetry, and recovery are inconsistent.
Level 2 — Documented	Teams create runbooks, dashboards, and release checklists, but capability is uneven and rarely exercised.
Level 3 — Integrated	Operational requirements, observability, release controls, incident response, and evidence are built into the lifecycle.
Level 4 — Risk-Proportionate	Readiness depth, exercises, independence, and runtime controls scale with authority, impact, and irreversibility.
Level 5 — Learning System	Production evidence, incidents, near misses, and operational exercises continuously improve architecture, controls, standards, and team capability.

Higher maturity does not mean more ceremony. Mature teams automate routine verification, keep evidence current, design smaller and safer changes, and focus human attention on uncertainty and consequence. Their operational confidence comes from tested capability and feedback, not from process volume.

15. The ETIS Position

ETIS treats operational readiness as the point at which engineering claims become accountable operating commitments. Repository-centered engineering preserves the record. Evidence-centered engineering supports the judgment. Engineering review challenges readiness claims. Governance defines authority and exception. Observability and postmortems return production truth to the lifecycle.

The ETIS position can be stated in seven propositions. First, operational readiness begins in requirements and architecture. Second, release success is not operational proof. Third, every important production claim should be observable. Fourth, ownership, intervention, and recovery must be explicit and exercised. Fifth, readiness must cover the full behavior-producing system, including models, prompts, policies, tools, data, and agents. Sixth, readiness depth should increase with authority, impact, uncertainty, and irreversibility. Seventh, production outcomes must change future engineering decisions.

ETIS Position

A trustworthy system is not merely capable of running. It is engineered so people can understand it, govern it, intervene when necessary, recover from failure, and improve it through evidence.

Implications for Leaders, Engineers, and Educators

For executives and engineering leaders, operational readiness is a business-control capability. Investments in development speed should be matched by investments in observability, recovery, platform controls, incident capability, and qualified ownership. AI increases this need because behavior can change through models, policies, prompts, context, and tool access outside traditional release patterns. Leaders should ask not only whether a system is delivering value, but whether the organization can detect, explain, constrain, and recover that value-producing behavior.

For engineers, operational thinking becomes part of design. The engineer must understand how the system will fail, what operators will see, how change will be introduced, what can be reversed, which dependencies matter, and what evidence will demonstrate health. The ability to write code remains important, but the ability to engineer diagnosable and recoverable behavior increasingly distinguishes professional work.

For educators, operational readiness should be taught before students reach production careers. Student projects should include run instructions, release baselines, telemetry, known limitations, defects, recovery assumptions, postmortems, and evidence-based readiness judgments. A working demonstration is valuable, but it is not a substitute for showing that another engineer could operate and understand the system.

Conclusion

Operational readiness is where software engineering accepts responsibility for production consequence. It asks whether a system can be observed, governed, supported, recovered, and sustained when assumptions fail and conditions change. That responsibility cannot be delegated to a late operations phase, because the capabilities needed in production are determined by requirements, architecture, implementation, delivery, and organizational design.

The AI era intensifies the challenge. Intelligent systems can change behavior dynamically, depend on external models and data, invoke tools, and act with increasing autonomy. Their operational state cannot be understood through infrastructure metrics alone. Organizations need visibility into decisions, context,

authority, cost, policy, human intervention, and outcome quality. They also need the ability to constrain or deactivate behavior that departs from intended use.

The future of operational engineering is not a larger launch checklist. It is a continuous readiness system: clear service expectations, observable behavior, controlled change, tested recovery, accountable ownership, risk-proportionate governance, and feedback that improves the next design. Organizations that build this capability can move quickly without confusing speed with preparedness. Those that do not will discover that production eventually tests every assumption they failed to examine.

Final Takeaway

Deployment places a system into production. Operational readiness prepares the organization to deserve that responsibility.

References

[1] Google Site Reliability Engineering. Production Readiness Review: Evolving SRE Engagement Model. https://sre.google/sre-book/evolving-sre-engagement-model/

[2] Amazon Web Services. Operational Excellence Pillar - AWS Well-Architected Framework. https://docs.aws.amazon.com/wellarchitected/latest/operational-excellence-pillar/welcome.html

[3] Microsoft. Azure Well-Architected Framework: Operational Excellence Design Principles. https://learn.microsoft.com/en-us/azure/well-architected/operational-excellence/principles

[4] OpenTelemetry. Observability Primer. https://opentelemetry.io/docs/concepts/observability-primer/

[5] OpenTelemetry. What Is OpenTelemetry? https://opentelemetry.io/docs/what-is-opentelemetry/

[6] DORA. Software Delivery Performance Metrics. https://dora.dev/guides/dora-metrics/

[7] Google Site Reliability Engineering. Launch Coordination Checklist and Canarying Releases. https://sre.google/sre-book/launch-checklist/ ; https://sre.google/workbook/canarying-releases/

[8] Cybersecurity and Infrastructure Security Agency. Federal Government Cybersecurity Incident and Vulnerability Response Playbooks. https://www.cisa.gov/resources-tools/resources/federal-government-cybersecurity-incident-and-vulnerability-response-playbooks

[9] Google Site Reliability Engineering Workbook. Incident Response and Postmortem Culture. https://sre.google/workbook/index/

[10] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework 1.0. https://www.nist.gov/itl/ai-risk-management-framework

[11] National Institute of Standards and Technology. AI RMF Playbook: Measure and Manage. https://airc.nist.gov/airmf-resources/playbook/

[12] OWASP GenAI Security Project. Top 10 for Agentic Applications and Agentic Security Guidance. https://genai.owasp.org/resource/owasp-top-10-for-agentic-applications-for-2026/

[13] Google Site Reliability Engineering. Accelerating SREs to On-Call and Beyond. https://sre.google/sre-book/accelerating-sre-on-call/

[14] International Organization for Standardization. ISO/IEC 42001:2023 Artificial Intelligence Management System. https://www.iso.org/standard/81230.html

[15] Google Site Reliability Engineering. The Site Reliability Workbook. https://sre.google/workbook/table-of-contents/

[16] Amazon Web Services. AWS Well-Architected Framework. https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html

[17] Microsoft. Azure Well-Architected Framework. https://learn.microsoft.com/en-us/azure/well-architected/

[18] DORA. State of AI-Assisted Software Development 2025. https://dora.dev/report

[19] OpenTelemetry. AI Agent Observability - Evolving Standards and Best Practices. https://opentelemetry.io/blog/2025/ai-agent-observability/

[20] Cybersecurity and Infrastructure Security Agency. AI Cybersecurity Collaboration Playbook. https://www.cisa.gov/resources-tools/resources/ai-cybersecurity-collaboration-playbook

[21] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. https://www.nist.gov/publications/artificial-intelligence-risk-management-framework-generative-artificial-intelligence

[22] Google Site Reliability Engineering. Communication and Collaboration in SRE. https://sre.google/sre-book/communication-and-collaboration/

[23] Microsoft. Azure Well-Architected Operational Excellence Checklist. https://learn.microsoft.com/en-us/azure/well-architected/operational-excellence/checklist

[24] DORA. Research Program and Core Model. https://dora.dev/research/

About the Author

William T. O'Connell, Ph.D. is an Adjunct Professor of Computer Science at Loyola University Chicago and the creator of the Engineering Trustworthy Intelligent Systems (ETIS) framework. His professional career spans more than four decades of software engineering, architecture, delivery, quality, and technology leadership, including service as an IBM Distinguished Engineer and CTO for IBM Consulting Quality and Delivery. His current work focuses on repository-centered engineering, engineering evidence, trustworthy intelligent systems, agentic software delivery, and the preparation of students and practicing engineers for increasingly AI-assisted engineering environments.

This white paper is part of the ETIS White Paper Series, which examines the engineering practices required to build, govern, review, operate, and sustain trustworthy software and intelligent systems.

Suggested citation: O'Connell, W. T. (2026). Operational Readiness: Engineering Systems That Can Be Observed, Recovered, Governed, and Sustained in Production. ETIS White Paper Series, WP-008, Version 1.0.