Aptlantis Analysis Standard (AAS): The Local Evaluation Pipeline Framework

1. The Strategic Pivot: From Quality Vibes to Credibility Engines

The transition from legacy repository management to a governed project ecosystem necessitates a fundamental shift in validation logic. Historically, assessing a project was a subjective "vibe check"—an exercise in guessing whether a repository appeared complete. This subjectivity imposes an "Interpretation Tax," where human operators and AI agents exhaust computational tokens and cognitive cycles attempting to discern the intent, maturity, and safety of a codebase. The Aptlantis Analysis Standard (AAS) replaces this ambiguity with a "local credibility engine." By anchoring all analysis on the **D:\ drive** for environmental resilience and portability, AAS ensures that validation is not a remote service but a local, persistent asset. Instead of asking "Did the repo pass?", AAS asks "What do we now understand about this project, and can we prove it?"**The AAS Mission:** To define a standard for local, explainable, and repeatable analysis pipelines that produce measurable evidence artifacts, ensuring every project remains understandable and recoverable throughout its lifecycle.

Cloud CI vs. AAS Local Credibility

Feature,Traditional Cloud CI/CD,AAS Local Credibility Engine
Primary Goal,Success/Failure (Green/Red),Understanding and Proof
Context,Generic Enterprise Defaults,Aptlantis Ecosystem Standards
Location,Remote/Hosted Services,Local-First / D:\ Drive Anchored
Output,Logs and Status Badges,Structured Evidence Artifacts
Agent Utility,Minimal (Opaque results),High (Agent Readiness Scoring)
This pivot ensures that as the workspace scales, the architectural integrity of projects like **Aegis** and **FileCabinet** is maintained through automated scrutiny rather than manual oversight.

2. The Anatomy of a Local Evaluation Pipeline

A project evaluation under the AAS is not a monolithic check. It is composed of "small evaluators," each designed to be narrow, explainable, and repeatable. This modular design prevents the "noisy" results typical of blind AI reviews and ensures every finding is linked to specific tool logic or ecosystem rules.The transformation from raw code to strategic recommendation follows the "Input-to-Recommendation" flow:

- **Input:** The raw project directory and core anchors (**project.manifest.toml** , **PROJECT.md**).
- **Collector:** Gathers data from deterministic tools (e.g., GitLeaks, compiler logs, directory crawlers).
- **Analyzer:** Processes gathered data against the requirements of the governing standard (e.g., **DRS** for desktop apps or **CTS** for CLI tools).
- **Evidence:** Normalizes the analyzer's output into structured, evidentiary facts.
- **Score:** Assigns a numerical value based on compliance and health metrics.

- **Report:** Generates human-readable summaries and machine-parseable artifacts.
- **Recommendation:** Suggests actionable next steps to reach the next maturity level.

Specialized Evaluator Roles

- **Secret Exposure:** Utilizes GitLeaks to ensure no sensitive credentials are at risk. *So What?* Prevents security breaches before a release is finalized.
- **Build Reality:** Executes environment-specific build commands—dotnet build for **FileCabinet** , cargo check for **CloneCratesio** , or python compileall for **AnalyzeProjects** . *So What?* Guards against "broken" repositories that cannot be implemented by others.
- **Documentation Anchor:** Verifies the presence of **README.md** , **ROADMAP.md** , and the **PROJECT.md** Identity Anchor. *So What?* Ensures project recoverability and prevents context loss during pauses.
- **Architecture Drift:** Compares the codebase against the **PPS (Project Proposal Standard)** "Design Boundaries" and the **PROJECT.md** Identity Anchor. *So What?* Prevents an agent or developer from violating established boundaries (e.g., introducing cloud dependencies into an offline-first project).These specialized evaluators serve as the foundation for the structured evidence that must be produced for every governed project.

3. The Credibility Model: Evidence Tiers and Scoring Logic

To maintain absolute trust between the architect and evaluation artifacts, AAS introduces "Evidence Levels." This tiered system distinguishes hard facts from heuristic suggestions, ensuring that the ecosystem separates deterministic truth from interpretation.

The Five Evidence Levels

1. **Observed:** Direct output from a trusted tool (e.g., a compiler error or a GitLeaks finding).
2. **Derived:** Facts calculated from multiple observed data points (e.g., total completion percentage).
3. **Inferred:** A reasonable interpretation based on existing facts (e.g., identifying a likely project stage).
4. **Suggested:** Recommendations generated by heuristics or local LLMs (e.g., suggesting a roadmap addition).
5. **Manual:** A judgment entered directly by a human operator.

Finding Schema Example

```
[[findings]]
id = "docs.missing-roadmap"
severity = "medium"
evidence_level = "observed"
status = "open"
message = "ROADMAP.md is missing from the project root."
source = "ROADMAP.md"
```

```
recommendation = "Add ROADMAP.md with current phase and next
milestones."
```

By categorizing findings, AAS ensures that automated agents and human maintainers can prioritize "Observed" failures while treating "Suggested" improvements as optional guidance. This structured environment defines the specific role of AI in the analysis process.

4. The Narrator Protocol: Defining the Role of Local LLMs

In the AAS, the boundary between "Interpretation" and "Authority" is strictly enforced. Local LLMs—hosted via **Ollama** —are integrated not as judges, but as narrators. The authority to declare a project "safe" or "ready" remains with deterministic tools; the LLM synthesizes these results into human-understandable context.

The 6-Step Governed Entry Protocol

To prevent "hallucination tax," LLM narration is strictly positioned as step 4 or 5 in the **Governed Entry Workflow** :

1. **Read Workspace Manifest:** Understand ecosystem inventory.
2. **Read Project Manifest:** Identify type and governing standard.
3. **Read PROJECT.md:** Absorb mission and design boundaries (Identity Anchor).
4. **Read Governing Standard (DRS/CTS): LLM Narration of requirements vs. current state.**
5. **Read Roadmap: LLM Analysis of phase alignment.**
6. **Begin Work.**

LLM Task Categorization

- **Good LLM Tasks (Narrative & Interpretation)**
- **Summarization:** Turning complex scan logs into high-level audit notes.
- **Gap Detection:** Identifying missing context between **PROJECT.md** and the file tree.
- **Roadmap Suggestions:** Proposing next steps based on current completion.
- **Bad LLM Tasks (Authority & Judgment)**
- **Declaring Security Safety:** LLMs must never replace deterministic scanners.
- **Approve Releases:** Release gates require deterministic checklist clearance.
- **Inventing Findings:** Creating errors without direct evidentiary support from tool output.This narrated data is preserved within the artifact architecture for long-term trend tracking.

5. The .evals Architecture: Artifact Management and Trend Tracking

The D:\.evals\ directory serves as the centralized audit hub for the entire Aptlantis workspace. By aggregating all health artifacts in one centralized location, the **Workspace Governance Standard (WGS)** enables cross-project analytics and historical trend tracking.

Centralized Directory Structure

```
D:\.evals\
```

```
├── FileCabinet\
│   ├── latest\
│   │   ├── build.eval.json
│   │   ├── docs.eval.json
│   │   └── release-readiness.eval.json
│   └── history\
│       └── 2026-06-01\
├── Aegis\
│   └── latest\
└── summary_trends.toml
```

Release Gates and DRS Compliance

These artifacts function as "Release Gates" for projects following the **DRS (Desktop Release Standard)** . A project is blocked from production if findings show open high-severity issues. Mandatory pass criteria for a **DRS-grade** project include:

- Verified compiler success (e.g., successful .NET 10 build).
- Zero high-severity secret exposures.
- Presence of all mandatory documentation anchors (**README.md** , **PROJECT.md** , **manifest**).
- **Artifact Integrity:** A valid and verified **SHA-256 or BLAKE3 hash** as specified in the release manifest.AAS provides the measurement layer that determines if these gates are cleared, preventing "file drops" from being classified as governed releases.

6. Ecosystem Synthesis: AAS as the Measurement Layer

AAS completes the "Aptlantis Trinity" of governance. While **WGS** defines the environment and **PPS** proposes the intent, AAS measures the reality of the project against its goals.

Standard-to-Evaluation Mapping

Standard,Governing Goal,AAS Verification Method
DRS,Desktop Release Integrity,"Verifies installer existence, SHA-256/BLAKE3 hashes, and release notes."
CTS,CLI Tool Predictability,"Validates exit codes, output schemas, and help text."
SFDS,Standard Maturity,"Audits for specification, examples, and Maturity Levels 0–4 compliance."
WGS,Workspace Organization,Ensures project registration and manifest consistency across the D/E drive split.

Agent Readiness Scoring

Beyond release readiness, AAS calculates an "Agent Readiness" score. This determines if a project is safe for AI-assisted edits. A project with high readiness (SFDS Level 3 or 4) possesses a clear **PROJECT.md** , a complete manifest, and passing build evaluators. High readiness scores signal to the architect that an agent can land in the repository and perform tasks without risking architectural drift.The Aptlantis philosophy is finalized in this measurement

layer: **The specification defines the rules, the examples demonstrate the rules, the validator proves the rules, and the evaluator (AAS) measures the rules.**