# TRP1 Week 2 Interim Report

**Project:** The Automaton Auditor (Digital Courtroom)
**Repository:** `/Users/gersumasfaw/Test/governance_swarm`
**Date:** 2026-02-24

## 1) Architecture Decisions Made So Far

### 1.1 Why Pydantic over plain dicts

We chose strict data contracts for evidence, judicial opinions, and final verdic
t outputs instead of passing loose dictionaries between nodes.

Decision:
- Use `BaseModel` for `Evidence`, `JudicialOpinion`, and verdict models.
- Keep graph state as `TypedDict` with reducer annotations for parallel branches
.

Why this is better for this challenge:
- Prevents silent schema drift between detectives, judges, and synthesis.
- Gives deterministic validation constraints for key fields (for example score r
anges and confidence bounds).
- Makes serialization to JSON/Markdown predictable for audit artifacts.

Current implementation references:
- `src/models.py` (typed models)
- `src/state.py` (state schema + reducers)

---

### 1.2 How AST parsing was structured

We used AST parsing in repository forensics to avoid brittle string-matching.

Decision:
- Parse Python source with `ast.parse(...)`.
- Traverse call nodes to evaluate graph wiring and inspect risky execution patte
rns.

Implemented protocol coverage:
- State structure checks
- Graph wiring checks (edge-call evidence)
- Security pattern checks (execution primitives)

Why this is better for this challenge:
- Structural verification is stronger than keyword/regex-only checks.
- Avoids executing untrusted target code while still extracting meaningful archi
tecture signals.
- Produces auditable evidence objects with rationale + confidence.

Current implementation references:
- `src/tools/repo_tools.py`

---

### 1.3 Sandboxing strategy for unknown repositories

The design isolates remote repository cloning in temporary directories and clean
s up in `finally` blocks.

Decision:
- Clone URL targets into `tempfile.TemporaryDirectory()`.

- Use `subprocess.run([...], check=True, capture_output=True)` with argument lis
ts (no shell interpolation).
- Always cleanup temporary clone handles.

Why this is better for this challenge:
- Prevents polluting the working tree during peer audits.
- Reduces shell-injection risk compared to raw shell strings.
- Keeps detective tooling reproducible across targets.

Current implementation references:
- `src/tools/repo_tools.py`

---

## 2) Known Gaps and Concrete Plan

### 2.1 Known gaps (current state)

1. Graph wiring does not yet include an explicit `EvidenceAggregator` fan-in syn
chronization node before judges.
2. Conditional edges for failure handling (evidence insufficiency, judge parse f
ailure, retry exhaustion) are not fully wired.
3. Final submission spec calls for `src/nodes/justice.py`; current synthesis log
ic lives in `src/nodes/supreme_court.py`.
4. Required helper interfaces are not yet exposed by spec names:
   - `analyze_graph_structure(path: str)`
   - `extract_git_history(path: str)`
   - `ingest_pdf(path: str)`
   - `extract_images_from_pdf(path: str)` (implementation required, execution op
tional)
5. Report rendering does not yet include per-criterion full judicial panel outpu
t (Prosecutor/Defense/TechLead arguments + citations) in required format.
6. Rubric depth is currently simplified (not yet aligned to full final scoring d
imensions).

---

### 2.2 Concrete implementation plan for Judicial Layer + Synthesis Engine

#### Phase A: Judicial Layer hardening
1. Add explicit judicial node contracts: one structured opinion per judge per cr
iterion.
2. Add parser-validation gate (`JudicialValidator`) after judge fan-in.
3. Add bounded retry policy when structured output validation fails.
4. Add citation validation: all cited evidence IDs must exist in evidence state.
5. Keep persona divergence checks; cap nuance score when collusion is detected.

Acceptance criteria:
- 3 valid judge outputs per criterion.
- No free-text-only opinion payloads pass validation.
- Retries are deterministic and bounded.

#### Phase B: Synthesis engine hardening
1. Move/finalize deterministic synthesis in `src/nodes/justice.py`.
2. Encode precedence rules:
   - Fact supremacy (evidence over opinion)
   - Security override (cap total score when confirmed negligence exists)
   - Dissent requirement (mandatory conflict explanation when variance is high)
3. Emit required report sections:
   - Executive Summary
   - Criterion Breakdown
   - Remediation Plan (file-level actions)

Acceptance criteria:
- No score is decided by opaque averaging alone.
- Conflict resolution is traceable to explicit deterministic rules.
- Output format is submission-ready.

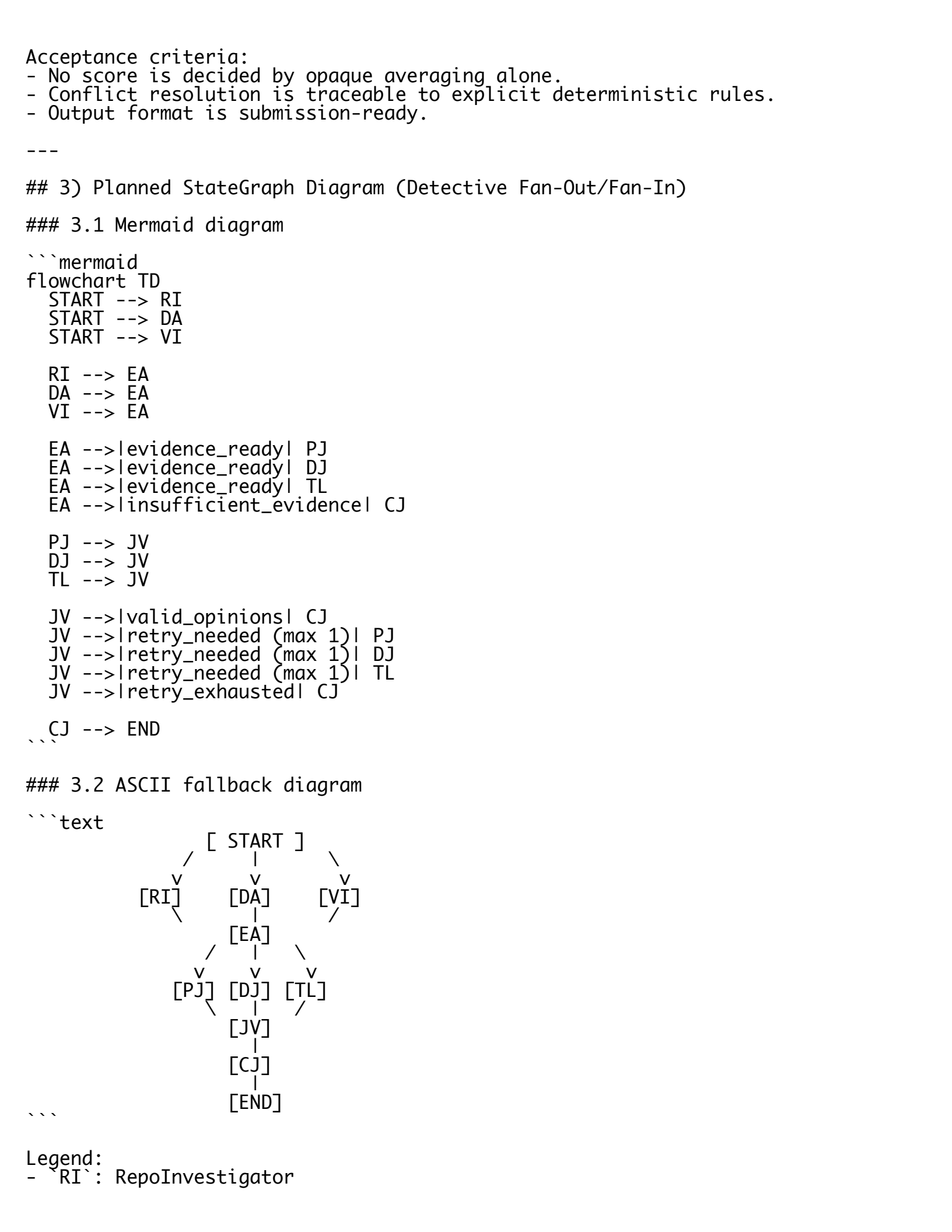
---

## 3) Planned StateGraph Diagram (Detective Fan-Out/Fan-In)

### 3.1 Mermaid diagram

```mermaid
flowchart TD
  START --> RI
  START --> DA
  START --> VI

  RI --> EA
  DA --> EA
  VI --> EA

  EA -->|evidence_ready| PJ
  EA -->|evidence_ready| DJ
  EA -->|evidence_ready| TL
  EA -->|insufficient_evidence| CJ

  PJ --> JV
  DJ --> JV
  TL --> JV

  JV -->|valid_opinions| CJ
  JV -->|retry_needed (max 1)| PJ
  JV -->|retry_needed (max 1)| DJ
  JV -->|retry_needed (max 1)| TL
  JV -->|retry_exhausted| CJ

  CJ --> END
```

### 3.2 ASCII fallback diagram




Legend:
- `RI`: RepoInvestigator

- `DA`: DocAnalyst
- `VI`: VisionInspector
- `EA`: EvidenceAggregator (fan-in barrier)
- `PJ/DJ/TL`: Prosecutor/Defense/TechLead
- `JV`: JudicialValidator (parse/citation gate + retry routing)
- `CJ`: ChiefJustice (deterministic synthesis)

---

## 4) Next Checkpoint Deliverables

Before final submission, this interim plan will be considered complete when:

1. Architecture and contracts are reflected in code structure and graph edges.
2. Judicial validator + retry behavior is observable in execution traces.
3. Deterministic synthesis rule table is implemented and tested on disagreement scenarios.
4. Audit output includes criterion-level judicial evidence and actionable remediation.