UNWIND — Design and Testing Document

MSSE Capstone Project

1. Project Overview

UNWIND is an interactive software system designed to train users to notice and interrupt habitual patterns of avoidance when uncomfortable internal sensations arise. The system detects when users leave direct experience (through story-telling, certainty-seeking, managing, etc.) and redirects them back to present-moment sensation.

The system is explicitly **not** a therapeutic tool. It does not soothe, teach, interpret, or provide insight. Its only function is to mirror escape patterns and maintain contact with sensation.

UNWIND is designed as a companion training tool for the book *Clear Seeing: From Bullshit Valley Back to Reality*. The book documents the conceptual framework; the app provides the behavioral practice. The system will be deployed on the author's website as a free resource for readers.

1.1 Origin and Development Path

The project emerged from direct experience using LLM-based conversational tools during the writing of the book. In that context, working with GPT was effective—the author was already focused on sensation and used the tool primarily for mirroring and clarifying language. The LLM's tendency to elaborate was useful for writing.

However, two problems became apparent:

1. **Users won't arrive pre-focused.** What worked for an author deep in sensation-based practice would not transfer to general users. Without existing focus, the LLM's engagement patterns would amplify escape rather than interrupt it.
2. **LLM drift was visible in the writing process itself.** Even while writing the book, the author observed the model softening language, adding reassurance, and subtly shifting toward therapeutic tone when the content touched difficult material.

An initial attempt was made to constrain LLM behavior through aggressive prompting (tested with Grok). The result was technically compliant but experientially terrible—mechanical, cold, and unusable. The constraints were correct; the execution was not.

This led to the current approach: build the constraint system first in pure Python, with no LLM dependency. Define every valid state, every allowed transition, every forbidden behavior. Once the contract is fully specified and tested, Phase 2 can reintroduce an LLM—not as the authority, but as a language generator operating within the validated constraint framework.

The Capstone deliverable represents Phase 1: a working, tested, deployable system that proves the interaction contract without relying on probabilistic generation.

2. Problem Statement

Conversational systems intended for reflection or self-inquiry reliably drift into unintended roles: therapist, teacher, coach, or authority. This drift undermines direct contact with experience and creates dependency on explanation or reassurance.

During early prototyping with LLM-based conversational agents, a specific failure mode was observed: the model would expand whatever pattern the user was already in. If the user was telling a story, the LLM elaborated the story. If the user claimed insight, the LLM validated the insight. If the user sought reassurance, the LLM provided it. This is structurally baked into how these models are trained—they optimize for engagement and continuity, not interruption.

This observation led to a fundamental design constraint: the system must be capable of **refusing to engage** with certain inputs, not just responding differently.

3. System Architecture

UNWIND is implemented as a deterministic, state-driven software system using the **State Machine Pattern**. This pattern was chosen because it:

- Makes all valid transitions explicit and auditable
- Prevents invalid state transitions by design
- Enables exhaustive testing of all paths
- Separates behavioral logic from presentation

3.1 Core Components

Pattern Classifier (classifier.py): A rule-based text classifier that identifies whether user input matches one of eight predefined escape patterns (Story, Certainty-Seeking, Problem-Solving, Claiming/Insight, Managing, Destination-Seeking, Floating, Unknown-Avoidance). Classification uses keyword matching with narrative markers, not probabilistic inference.

State Machine (machine.py): A finite state machine governing all session transitions including entry routing, orientation flow, Mirror Mode, sensation tree navigation, dense/spacious paths, and session termination. The machine maintains session state and enforces valid transitions.

Sensation Tree (sensation_tree.py): A hierarchical menu structure for refining sensation descriptions through forced-choice selection (domain → refinement → one-word). This replaces free-form description to avoid triggering escape pattern detection on legitimate sensation reports.

Template System (templates.py): All system prompts are predefined templates. No dynamic text generation occurs. This ensures behavioral consistency and prevents drift.

3.2 Constraint Engine

Hard constraints define what the system cannot do, enforced at the controller level:

- No soothing or calming language
- No teaching, explanation, or insight provision
- No interpretation of user experience
- No encouragement or praise
- No advancement of process without user-initiated transition

4. Key Design Decisions

4.1 Deterministic Control Over Generative Flexibility

Early design considered integrating an LLM for natural language generation. This was intentionally deferred for three reasons:

3. **Behavioral drift:** LLMs are trained on conversational data that rewards engagement and continuity. In domains requiring interruption rather than elaboration, this creates a fundamental misalignment.
4. **Architectural complexity:** A dual-engine architecture (LLM + constraint controller) would require the controller to override LLM outputs, introducing latency, cost, and failure modes.
5. **Scope validation:** The Capstone objective was to validate that the interaction contract itself works, independent of language generation quality.

The architecture preserves a future extension point where an LLM could be introduced behind the existing controller, with the deterministic system remaining the authority for allowed transitions.

4.2 Forced-Choice Menus for Sensation Refinement

A specific failure mode was discovered during prototyping: when users were asked to describe sensation in free text, the LLM could not reliably distinguish "language as report" from "language as escape." A user saying "tight chest" might be genuinely reporting sensation or constructing a story about sensation. The LLM's defensive response was to restart the loop—returning to "What's in your central view?"—even when the user was already in contact with sensation.

The insight was that once the user has entered sensation territory, the interaction rules should change. At that point, the system no longer needs to classify free text—it can guide the user through structured refinement. The solution was forced-choice menus: users select from predefined sensation domains (Pressure, Temperature, Movement, etc.) and refinements (Tight, Expanding, Pulsing, etc.), then proceed to the dense/spacious fork where actual learning occurs.

This design separates the **detection phase** (where escape patterns must be caught) from the **contact phase** (where structured guidance is appropriate). The forced menus mark the boundary.

4.3 Attunement vs. Purity

Early testing with aggressive constraints (Grok) produced a system that was technically correct but experientially hostile. Users felt dismissed rather than redirected. The constraints were right; the experience was unusable.

The solution was minimal attunement: on first detection of each escape pattern, the system provides a one-sentence acknowledgment ("That's leaving sensation. It moves attention away from what's actually here."). On subsequent detections, only the pattern name appears.

This deliberately breaks constraint purity to preserve usability. A system that maintains perfect boundaries but loses the user accomplishes nothing. The design tradeoff is explicit: **attunement first time, brevity thereafter**.

5. Deployment Options and Cost Analysis

UNWIND is deployed as a static web application requiring no server-side processing.

Option	Description	Cost
Static hosting	Netlify, Vercel, GitHub Pages	$0/month (free tier)
Self-hosted	Any web server (Apache, Nginx)	Hosting costs only
Cloud (future)	With LLM integration	~$0.01-0.05/session

Recommendation: Static hosting (Netlify/Vercel free tier) for current version. Zero ongoing cost, global CDN, automatic HTTPS. If LLM integration is added, move to serverless functions with pay-per-use pricing.

6. Testing Strategy

Testing focused on verifying deterministic correctness, not subjective outcomes. The system must behave identically given identical inputs.

6.1 Test Categories

Category	Tests	Purpose
Story Detection	6	Narrative markers, emotion+context, venting patterns
Escape Patterns	14	All 8 patterns correctly classified

Crisis Detection	3	Safety keywords trigger immediate exit
Entry Routing	4	New vs returning user flow
Orientation Flow	4	13 screens, safety gate enforcement
Mirror Mode	7	Pattern mirroring, sensation tree entry
Sensation Tree	5	Domain/refinement selection by number/name
Clench/Fork/Dense Paths	10	Branching logic, layer progression
Session End	7	Clean termination, offapp handoff

6.2 Testing Tools and Rationale

pytest was selected for unit testing because it:

- Provides clear pass/fail assertions with detailed failure output
- Supports coverage reporting (pytest-cov) for identifying untested paths
- Enables test organization by class/category
- Is the Python community standard, well-documented

Scenario scripts complement unit tests by simulating complete user sessions end-to-end, validating that the state machine produces expected outputs for realistic interaction sequences.

6.3 Test Results

- **60 automated tests**
- **0 failures**
- **72% code coverage** across controller and language modules

To run tests:

```
cd unwind_v2 && python -m pytest tests/ -v --cov=controller --cov=language
```

7. Limitations and Future Work

Current limitations:

- No free-form conversational language (all responses are templates)
- No personalization or learning across sessions
- No mobile-native interface (web-only)

Planned future work:

- Optional LLM integration for natural language generation, constrained by existing state machine
- Session logging for pattern analysis (opt-in)

- Mobile app wrapper

8. Conclusion

UNWIND demonstrates that meaningful interactive systems can be built by prioritizing constraint over capability. The project validates a deterministic architecture capable of maintaining behavioral integrity without drift, providing a stable foundation for future extensions.

9. Phase Two Vision — Attunement, Agency, and Sustainability

Phase One validated the core mechanism: attention returns to sensation when narrative, explanation, and management are not followed. The constraint-first system proved effective as a training tool.

However, Phase One also surfaced a central design tension:

Purity versus attunement.

Systems that enforce purity too rigidly are technically correct but experientially alienating. Systems that prioritize attunement without constraint feel supportive but reliably drift into narrative reinforcement. Phase Two is not about choosing one side, but deliberately managing the tension between them.

This can be understood through a navigation model.

An aircraft is constantly blown off course by wind. Attempting to hold a perfectly rigid heading risks structural failure; allowing unlimited drift never reaches the destination. Effective navigation permits deviation while continuously correcting toward the heading.

Phase Two adopts this model.

The destination (heading) remains unchanged: sustained orientation toward direct bodily sensation. Deviation is permitted, but never without ongoing correction.

9.1 Phase Two Direction: Agentic Control with Adaptive Pressure

Phase Two introduces an **agentic layer** whose role is not to replace the constraint logic, but to **mediate how it is applied**.

The agent has a stable, non-negotiable goal:

Maintain orientation toward sensation without becoming a co-author of narrative.

The agent is allowed limited attunement behaviors:

- Brief mirroring of user language
- One-sentence acknowledgment of emotional tone
- Transitional language that preserves relational continuity

The agent is explicitly **not** allowed to:

- Ask "why" questions about story content
- Introduce new interpretations
- Offer solutions, advice, or meaning
- Validate narrative truth claims
- Extend or elaborate plot ("what happened next?")

This is not a conversational agent in the therapeutic sense. It is a **guided navigator**.

Crucially, Phase One constraints remain the **heading**, not a leash. The agent may bend, but it may not redefine success.

9.2 Critic / Governor: Preventing Drift Without Rigidity

Phase Two avoids hard turn-count limits or brittle rules. Instead, it introduces a **self-checking critic function**.

This critic is not a second full agent with goals of its own. It is a lightweight evaluative step that answers a small number of structured questions after each response:

- Did the last response advance narrative content?
- Did it introduce interpretation, advice, or meaning?
- Did it validate the story as true or justified?
- Did it move attention toward or away from sensation?

This can be implemented in two ways:

1. As a second, low-token LLM pass (critic)
2. As a structured self-evaluation within a single agent pass

Because the critic only evaluates behavior (not generating language), token cost is minimal and predictable.

If the critic detects excessive narrative momentum, **pressure escalates**:

- Mirroring becomes shorter
- Acknowledgment is reduced
- Redirection becomes more explicit

If the critic detects readiness (lower arousal, shorter narrative loops), pressure relaxes.

This creates **adaptive pressure without fixed turn counts**, allowing conversations to ebb and flow while still converging.

The agent is not punished for deviation; it is gently steered back.

9.3 Attunement vs. Co-Authoring: The Key Boundary

The primary failure mode is not "allowing story."
It is **joining the story**.

Attunement is permitted as long as it does not:

- Add new narrative material
- Strengthen identity positions
- Provide moral or factual validation
- Shift focus away from bodily experience

Mirroring is treated as an attunement tool, not an insight engine. Its purpose is to help the user feel seen long enough to remain in the container.

The system explicitly prefers **retention over purity**, as long as orientation is preserved.

A perfectly pure system that users abandon is ineffective.

9.4 Economics and Usage Model

Phase Two is **text-first** to keep costs low and behavior predictable.

Pricing:

- $8/month
- $72/year

Access model:

- Orientation available without signup
- Account signup grants up to 4 hours of use
- Cancellation permitted within that window

Usage policy:

- Up to 60 hours per month of active use included
- No usage-based charges
- A hard cap exists solely to prevent extreme or automated abuse

At this cap, variable costs remain well below a 2× margin, ensuring sustainability without incentivizing excessive engagement.

The system is designed to support brief, repeated use — not prolonged sessions.

9.5 Summary

Phase Two is not an expansion in features.

It is a **rebalancing**:

- Constraint remains the heading
- Agency manages pressure
- Attunement preserves relationship
- A critic prevents narrative co-authoring
- Economics remain simple and defensible

The result is a system that can bend without breaking, retain users without drifting, and remain aligned with the original mechanism.