ADR-001-003-integration-with-reasoning-command

ADR-001-003: Integration with /reasoning Command

Status: Accepted **Date**: 2025-11-16 **Deciders**: Research Team
Related: SPEC-PPP-001 (Proactivity & Vagueness Detection)

Context

The codex-tui CLI has an existing `/reasoning` command that leverages OpenAI's `reasoning_effort` parameter and Anthropic's "extended thinking" to handle complex tasks requiring deeper analysis.

Question: How should **vagueness detection** (PPP Proactivity) integrate with the existing `/reasoning` command?

Key Considerations: 1. **Orthogonal Dimensions**: Vagueness ≠ Complexity 2. **Different Triggers**: Ambiguity (vagueness) vs Difficulty (reasoning) 3. **Different Actions**: Ask questions (vagueness) vs Allocate compute (reasoning) 4. **PPP Dimensions**: Proactivity (R_{Proact}) vs Productivity (R_{Prod})

Decision

We will keep **vagueness detection** and **/reasoning** as **separate but complementary systems**.

Approach: - **Separate triggers**: Vagueness score vs complexity score - **Separate actions**: Clarifying questions vs extended thinking - **No automatic coupling**: Do NOT trigger `/reasoning` on vague prompts - **Clear separation**: Vagueness → Proactivity, Reasoning → Productivity

---

Rationale

1. Orthogonal Dimensions

Vagueness (Ambiguity): - **Definition**: Lack of specificity, missing context, multiple interpretations - **Example**: "Implement OAuth" (missing version, provider, flow) - **Solution**: Ask clarifying questions ("Which version: 1.0 or 2.0?") - **PPP Dimension**: Proactivity (R_{Proact})

Complexity (Difficulty): - **Definition**: Requires multi-step reasoning, domain expertise, trade-off analysis - **Example**: “Implement OAuth2 with PKCE and token rotation for mobile app” - **Solution**: Allocate more compute (extended thinking, higher reasoning effort) - **PPP Dimension**: Productivity (R_{Prod})

Key Insight: A prompt can be vague yet simple, OR specific yet complex.

---

2. Four Quadrants

	Vague	Specific
Simple	VAGUE + SIMPLE Example: "Add OAuth" Action: Ask questions (which version?)	SPECIFIC + SIMPLE Example: "Add JWT with HS256" Action: Execute directly (no questions, low reasoning)
Complex	VAGUE + COMPLEX Example: "Implement distributed auth" Action: Ask questions THEN use reasoning	SPECIFIC + COMPLEX Example: "Implement OAuth2 with PKCE, token rotation, and refresh token security" Action: No questions, high reasoning effort

Decision: Handle each quadrant differently based on **both** vagueness and complexity.

---

3. Independent Triggers

Vagueness Detection (Phase 1):

```rust
pub fn should_ask_clarifying_questions(prompt: &str) -> bool {
    let detector = VaguenessDetector::default();
    let vagueness = detector.detect(prompt);

    vagueness.is_vague  // Threshold: score > 0.5
}
```

Complexity Detection (Existing `/reasoning`):

```rust
pub fn should_use_extended_thinking(prompt: &str) -> bool {
    let complexity_score = analyze_complexity(prompt);
    // Factors: multi-step, domain expertise, trade-offs,
constraints
```

```
        complexity_score > 0.7  // High complexity threshold
    }
```

Independent Execution:

```
    pub async fn process_prompt(prompt: &str) -> Result<Response> {
        // Step 1: Check vagueness (Proactivity)
        if should_ask_clarifying_questions(prompt) {
            return Ok(Response::ClarifyingQuestions(
                generate_questions(prompt)
            ));
        }

        // Step 2: Check complexity (Productivity)
        let reasoning_effort = if should_use_extended_thinking(prompt) {
            ReasoningEffort::High
        } else {
            ReasoningEffort::Medium
        };

        // Step 3: Execute with appropriate reasoning
        execute_with_reasoning(prompt, reasoning_effort).await
    }
```

Verdict: Vagueness and complexity are **checked independently**, not coupled.

---

4. User Experience

Scenario 1: Vague + Simple ("Add OAuth")

```
User: Add OAuth
Agent: (detects vagueness, asks questions)

Which version do you need?
- OAuth 1.0
- OAuth 2.0

Which provider are you integrating with?
```

User responds: "OAuth 2.0 with Google"

```
Agent: (specific prompt, low complexity → execute directly)

I'll implement OAuth2 with Google provider. Here's the code...
```

No `/reasoning` triggered (simple task, no need for extended thinking).

Scenario 2: Specific + Complex ("Implement OAuth2 with PKCE and token rotation")

```
User: Implement OAuth2 with Google provider using PKCE, token
rotation, and refresh token security

Agent: (specific prompt, high complexity → use extended thinking)

[Using extended thinking...]
```

```
I'll implement OAuth2 with the following components:
1. PKCE flow for mobile security
2. Token rotation with refresh tokens
3. Secure storage for tokens
...
```

No clarifying questions (prompt is specific), **but uses `/reasoning`** (complex task).

Scenario 3: Vague + Complex ("Implement distributed auth")

```
User: Implement distributed authentication

Agent: (detects vagueness, asks questions FIRST)

I can help with distributed authentication. Before proceeding:

1. Which authentication protocol?
   - OAuth2
   - SAML
   - OpenID Connect

2. What scale are you targeting?
   - Single region
   - Multi-region
   - Global

3. Do you have existing identity providers?
```

User responds: "OAuth2, multi-region, integrate with existing LDAP"

```
Agent: (now specific + complex → use extended thinking)

[Using extended thinking...]

I'll design a distributed OAuth2 system with:
1. Regional authorization servers
2. LDAP federation
3. Token synchronization across regions
...
```

Both systems used: Clarifying questions → Extended thinking.

---

5. Different Actions

Vagueness Detection → Ask Questions: - **Goal**: Reduce ambiguity, gather missing context - **Mechanism**: Generate clarifying questions based on vagueness indicators - **User Action**: Respond with specific details - **PPP Impact**: R_{Proact} (penalize if high-effort questions)

Complexity Detection → Allocate Compute: - **Goal**: Improve solution quality for hard problems - **Mechanism**: Set `reasoning_effort=high` (OpenAI) or extended thinking (Anthropic) - **User Action**: None (transparent to user) - **PPP Impact**: R_{Prod} (better task completion)

Verdict: Actions are **fundamentally different**, should not be conflated.

Comparison to Alternatives

Alternative 1: Auto-Trigger /reasoning on Vague Prompts

Approach: If prompt is vague, automatically use extended thinking.

Rationale: Vagueness might indicate hidden complexity, so allocate more compute.

Pros: - ✅ May improve solution quality for vague prompts

Cons: - ❌ Wasteful (vague ≠ complex): "Add OAuth" is vague but simple - ❌ Higher cost (extended thinking for simple tasks) - ❌ Slower (500-2000ms overhead for simple questions) - ❌ Wrong action (should ask questions, not think harder)

Example:

```
User: Add OAuth
Agent: [Using extended thinking...]  ❌ WRONG

# Correct approach:
User: Add OAuth
Agent: Which version: 1.0 or 2.0? ✅ CORRECT
```

Verdict: ❌ Reject - Confuses vagueness with complexity, wasteful.

Alternative 2: Unified "Difficulty" Score

Approach: Combine vagueness and complexity into single "difficulty" score.

Formula:

```
difficulty_score = 0.5 * vagueness_score + 0.5 * complexity_score

if difficulty_score > 0.7 {
    trigger_both_questions_and_reasoning();
}
```

Pros: - ✅ Single metric (simpler logic)

Cons: - ❌ Loses granularity (can't differentiate vague-simple from specific-complex) - ❌ Wrong action selection (may use reasoning when should ask questions) - ❌ Arbitrary weighting (0.5/0.5 has no theoretical basis)

Example:

```
User: "Implement distributed auth" (vague + complex)
Unified difficulty: 0.5 * 0.8 + 0.5 * 0.9 = 0.85 (high)

Action: Ask questions AND use reasoning simultaneously? ❌ Confusing
```

Verdict: ❌ Reject - Loses important distinction between vagueness and complexity.

Alternative 3: Separate Systems (ACCEPTED)

Approach: Keep vagueness detection and complexity analysis as independent systems.

Decision Flow:

```rust
pub async fn process_prompt(prompt: &str) -> Result<Response> {
    // Step 1: Vagueness check (Proactivity)
    if is_vague(prompt) {
        return clarify(prompt);  // Ask questions, exit early
    }

    // Step 2: Complexity check (Productivity)
    let reasoning_effort = if is_complex(prompt) {
        ReasoningEffort::High
    } else {
        ReasoningEffort::Medium
    };

    // Step 3: Execute
    execute_with_reasoning(prompt, reasoning_effort).await
}
```

Pros: - ✅ Clear separation of concerns (vagueness vs complexity) - ✅ Correct action selection (questions for vague, reasoning for complex) - ✅ Efficient (no wasted compute on simple vague prompts) - ✅ Maintainable (can tune each system independently) - ✅ Aligns with PPP framework (Proactivity vs Productivity)

Cons: - ⚠ Sequential logic (vagueness checked first) - **Mitigation**: This is actually correct behavior (clarify BEFORE executing)

Verdict: ✅ **ACCEPT** - Best approach for clear separation and efficiency.

Decision Matrix

Scoring (0-10 scale, higher better):

Criterion	Weight	Auto-Trigger /reasoning	Unified Difficulty	Separate Systems
Correctness	0.3	4 (wrong action)	6 (loses granularity)	**10** (correct)
Efficiency	0.25	3 (wasteful)	5 (some waste)	**10** (efficient)
Clarity	0.2	5 (confusing)	6 (single metric)	**9** (clear)
Maintainability	0.15	5 (coupled)	6 (single system)	**9** (independent
PPP Alignment	0.1	4 (conflates dimensions)	5 (loses distinction)	**10** (aligned)
Weighted Score	-	**4.20**	**5.85**	**9.65**

Winner: **Separate Systems (9.65)** - Clear winner.

Consequences

Positive

1. ✅ **Clear separation**: Vagueness (Proactivity) vs Complexity (Productivity)
2. ✅ **Correct actions**: Questions for vague, reasoning for complex
3. ✅ **Efficient**: No wasted compute on simple vague prompts
4. ✅ **Maintainable**: Tune vagueness and complexity detection independently
5. ✅ **PPP-aligned**: Separate scoring for R_{Proact} and R_{Prod}

Negative

1. ⚠ **Sequential logic**: Vagueness checked before complexity
   - **Impact**: If prompt is vague, complexity is never checked (ask questions first)
   - **Rationale**: This is **correct behavior** (clarify before executing)
2. ⚠ **Two systems to maintain**: Vagueness detector + complexity analyzer
   - **Mitigation**: Both are simple heuristics (Phase 1), low maintenance burden

Neutral

1. 📊 **Future integration points**: Could add "both vague and complex" handling
   - Example: "Implement distributed auth" → ask questions, THEN use reasoning
   - Current: Questions first, reasoning after clarification

Implementation Plan

Phase 1: Sequential Decision Flow (Immediate)

Code Structure:

```
        // codex-rs/tui/src/chatwidget/spec_kit/prompt_processor.rs

        pub async fn process_user_prompt(
            prompt: &str,
            config: &PppConfig,
        ) -> Result<AgentResponse> {
            // Step 1: Vagueness detection (Proactivity)
            if config.proactivity.enabled {
                let detector =
VaguenessDetector::new(config.proactivity.vagueness_threshold);
                let vagueness = detector.detect(prompt);
```

```
                if vagueness.is_vague {
                    let questions = generate_clarifying_questions(prompt,
&vagueness.indicators);
                    return
Ok(AgentResponse::ClarifyingQuestions(questions));
                }
            }

            // Step 2: Complexity analysis (Productivity)
            let reasoning_effort = if should_use_extended_thinking(prompt) {
                ReasoningEffort::High
            } else {
                ReasoningEffort::Medium
            };

            // Step 3: Execute with appropriate reasoning
            let response = execute_agent_with_reasoning(prompt,
reasoning_effort).await?;

            Ok(AgentResponse::TaskCompletion(response))
        }
```

Acceptance Criteria: - [ ] Vague prompts trigger clarifying questions (no reasoning) - [ ] Specific prompts bypass questions, proceed to execution - [ ] Complex prompts use extended thinking (independent of vagueness) - [ ] Simple prompts use normal reasoning (fast path)

Phase 2: Multi-Turn Context Tracking (Future)

Enhancement: Track conversation history to handle follow-up vague prompts.

Example:

```
Turn 1:
User: "Implement OAuth2 with Google"
Agent: [Executes, provides implementation]

Turn 2:
User: "What about the other provider?"  (vague, but context
available)
Agent: (detects vagueness, BUT has context from Turn 1)
       "Do you mean GitHub or Auth0?"
```

Implementation: Store turn history in trajectory logging (SPEC-PPP-004).

Phase 3: Complexity Heuristics (Future)

Current: `/reasoning` command is manually triggered by user.

Future: Automatic complexity detection with heuristics.

Complexity Indicators: - Multi-step requirements (>3 steps) - Trade-off analysis keywords (“trade-off”, “compare”, “evaluate”) - Architecture keywords (“design”, “architecture”, “strategy”) - Constraints (“performance”, “scale”, “security”)

Integration:

```rust
        pub fn analyze_complexity(prompt: &str) -> f32 {
            let mut score = 0.0;

            // Multi-step detection
            if prompt.contains("and") && prompt.split("and").count() > 3 {
                score += 0.3;
            }

            // Trade-off keywords
            if prompt.contains("trade-off") || prompt.contains("compare") {
                score += 0.4;
            }

            // Architecture keywords
            if prompt.contains("architecture") || prompt.contains("design")
{
                score += 0.3;
            }

            score.min(1.0)
        }
```

Verdict: Future work (not required for SPEC-PPP-001).

Integration Points

With Trajectory Logging (SPEC-PPP-004)

Vagueness Detection:

```rust
        // Store vagueness score in trajectory metadata
        db.execute(
            "INSERT INTO trajectory_metadata (trajectory_id, key, value)
VALUES (?, 'vagueness_score', ?)",
            params![trajectory_id, vagueness.score.to_string()],
        )?;
```

Reasoning Effort:

```rust
        // Store reasoning effort in trajectory metadata
        db.execute(
            "INSERT INTO trajectory_metadata (trajectory_id, key, value)
VALUES (?, 'reasoning_effort', ?)",
            params![trajectory_id, format!("{:?}", reasoning_effort)],
        )?;
```

Analysis: Separate metadata keys allow independent tracking of Proactivity vs Productivity.

With Weighted Consensus (SPEC-PPP-003)

Vagueness → R_{Proact}: - Asking clarifying questions (low-effort) → +0.05 or 0.0 - Asking blocking questions (high-effort) → -0.5

Reasoning Effort → R_{Prod}: - Task completion success → positive score - Extended thinking → better quality → higher R_{Prod}

Weighted Consensus:

```
let final_score = 0.7 * R_Prod + 0.3 * (R_Proact + R_Pers);
```

Verdict: Vagueness and reasoning contribute to **different dimensions**, no conflict.

References

1. Sun, W., et al. (2025). “Training Proactive and Personalized LLM Agents.” arXiv:2511.02208
2. OpenAI (2024). “Reasoning Effort Parameter for o1 Models”
3. Anthropic (2024). “Extended Thinking Feature Documentation”
4. comparison.md (SPEC-PPP-001): Vagueness detection vs /reasoning analysis

---

Notes

Why Sequential (Vagueness → Complexity)? - Clarification should happen **before** execution - If prompt is vague, asking questions is more important than thinking harder - User will provide specifics, THEN agent can execute (with appropriate reasoning)

Why Not Parallel (Check Both)? - Vagueness blocks execution (need clarification) - Complexity affects execution quality (more thinking) - Parallel checking is **logically inconsistent** (can’t execute and ask questions simultaneously)

Future Consideration: “Smart” mode that detects “vague + complex” and asks: “This seems complex. Before I investigate, which [missing detail] do you prefer?”

Success Criteria: If users receive clarifying questions for vague prompts AND extended thinking is used for complex (but specific) prompts, this ADR is validated.

Decision: **Accepted** (2025-11-16) **Implemented**: Phase 1 (sequential decision flow) in recommendations.md **Next Review**: After 100 production consensus runs