CLAUDE'S CLARIFICATION FRAMEWORK V2

Instructions for processing voice-dictated PRDs (TRUE Method)

MISSION

Transform raw voice dictation into agent-ready PRD through:

1. Intelligent parsing of natural speech
2. Systematic gap identification
3. Targeted clarifying questions
4. Real-time PRD assembly

Goal: Minimize back-and-forth while maximizing requirement completeness.

PHASE 1: INITIAL PARSE

Extract and Categorize

Parse the voice transcript into these buckets:

```
DECLARED:
- What user explicitly stated
- Concrete technical decisions
- Named integrations/tools
- Specific constraints

IMPLIED:
- Unstated but logically necessary
- Industry standard assumptions
- Common patterns for this stack

MISSING:
- Critical gaps that block implementation
- Ambiguous statements needing clarification
- Contradictions to resolve

RISKS:
- Statements that suggest complexity
- Integration points without auth details
- Vague scope boundaries
- Edge cases not considered
```

Triage by Criticality

ASK IMMEDIATELY (Blockers):

- File paths / repository structure
- Authentication mechanisms
- API keys / credentials location
- Database schemas

- Environment setup

ASK SECOND (Implementation Details):

- Error handling specifics
- UI component choices
- Caching strategies
- Logging approaches

PROPOSE SMART DEFAULTS (Non-Blockers):

- Testing frameworks
- Documentation format
- Code style preferences

PHASE 2: QUESTION CONSTRUCTION

Rules for Asking Questions

❌ **DON'T ASK:**

```
"What are your requirements?" (too broad)
"How should authentication work?" (already specified)
"What database do you want?" (ask ONLY if truly not mentioned)
```

✅ **DO ASK:**

```
"You mentioned Stammer AI API but not the auth method. Is it API key, OAuth, or JWT?"
"Where should I create the /dashboard folder - in /app or /pages?"
"Should failed API calls retry automatically or require manual refresh?"
```

Question Pattern Template

```
CONTEXT + GAP + SMART OPTIONS

"You mentioned [what they said] but didn't specify [what's missing].
Should I assume [smart default] or do you want [alternative]?"
```

Examples of GOOD Questions

Authentication:

> *"You mentioned user login but not the mechanism. OAuth with Google, magic links via email, or username/password?"*

File Structure:

> *"Where should the dashboard components live? /app/dashboard or /pages/dashboard? And should I use Next.js 13+ app router or pages router?"*

API Integration:

> *"For the Stammer API calls, what's the error handling? Retry 3 times then fail, or show cached data immediately?"*

Data Storage:

> *"You said 'store user preferences' - in Supabase table, localStorage, or both (with sync)?"*

PHASE 3: PROGRESSIVE QUESTIONING

Ask in Dependency Order

WRONG ORDER:

```
1. What icon library?
2. Where's the API key?
3. What's the deployment strategy?
```

Jumping around wastes time

RIGHT ORDER:

```
1. Where's the code going? (file paths)
2. How does auth work? (access gates)
3. Where are credentials? (API keys)
4. What's the data flow? (architecture)
5. How should errors show? (UX)
6. What icons/styling? (polish)
```

One Question at a Time

❌ **BAD:**

> *"What about authentication and where should the files go and do you want TypeScript and what about testing?"*

✅ **GOOD:**

> *"Let's start with the code location. Should I create a new /dashboard folder under /app, or does this go somewhere else?"*

Wait for answer. Then next question.

PHASE 4: SMART DEFAULTS

When to Propose vs Ask

PROPOSE when:

- Industry standard practice
- User's previous projects set pattern
- Technical best practice
- Low-risk decision

ASK when:

- Business logic decision
- Integration point specification
- Data handling/security
- Anything that could block autonomous execution

Default Proposal Template

```
"I didn't see [X] mentioned. I'll assume [smart default]
unless you say otherwise. Sound good?"
```

Examples of Smart Defaults

Testing:

> *"I'll set up Vitest for unit tests since you're using Vite. Shout if you prefer Jest."*

Styling:

> *"For the dashboard, I'll use Shadcn's default theme unless you have custom colors."*

Error Pages:

> *"I'll create a generic 500 error page. We can customize it later if needed."*

Documentation:

> *"I'll update the README with setup instructions and API integration notes."*

PHASE 5: CONFIRMATION PATTERNS

Readback for Validation

After gathering clarifications, do a structured readback:

```
"Here's what I understand:

PROJECT: [name and purpose]

STACK:
- [framework] [version]
- [key packages]

LOCATION: [file path]

INTEGRATIONS:
- [service]: [auth method]
- [service]: [API pattern]

KEY FEATURES:
1. [feature] - [acceptance criteria]
2. [feature] - [acceptance criteria]

CONSTRAINTS:
- [must not do X]
- [performance requirement]

Does this match your intent? Anything I misunderstood?"
```

Wait for confirmation before generating final PRD.

PHASE 6: PRD ASSEMBLY

Structure Final Output

Generate the agent-ready JSON following this order:

1. **Project metadata** - name, type, goals
2. **Technical spec** - stack, paths, dependencies
3. **User stories** - ATOMIC, testable, ordered by dependency
4. **Data flow** - inputs, transforms, outputs
5. **Error handling** - scenarios and responses
6. **Testing** - unit, integration, manual
7. **Success criteria** - functional, technical, quality
8. **Constraints** - must-nots, performance, scale
9. **Safety limits** - iteration caps, timeouts

User Story Quality Checklist - CRITICAL FOR TRUE METHOD

Each story MUST:

- ✅ Be completable in ONE fresh Claude Code session (5-15 min max)
- ✅ Have clear, atomic scope (one feature/component only)
- ✅ Include all acceptance criteria in GIVEN/WHEN/THEN format
- ✅ Specify exact file targets
- ✅ Call out dependencies explicitly
- ✅ Provide technical implementation hints

WHY ATOMICITY MATTERS: Each user story executes in a FRESH Claude Code session. If a story is too complex, context rot degrades performance mid-execution. Keep stories small enough to complete in one session.

❌ **BAD STORY (too broad):**

```
{
  "title": "Build complete authentication system",
  "acceptanceCriteria": ["Users can sign up, log in, reset password, and manage profile"]
}
```

This would require multiple sessions and accumulate context rot

✅ **GOOD STORY (atomic):**

```
{
  "title": "Implement OAuth login with Google",
  "acceptanceCriteria": [
    "GIVEN user clicks 'Login with Google' WHEN OAuth flow completes THEN user is redirected
to /dashboard with valid JWT",
    "AND JWT is stored in httpOnly cookie with 7-day expiry",
    "AND failed OAuth redirects to /login with error message"
  ],
  "fileTargets": ["/app/api/auth/google/route.ts", "/lib/auth.ts"],
  "dependencies": ["US-000: Setup Supabase project"],
  "technicalHints": [
    "Use @supabase/auth-helpers-nextjs",
    "Configure OAuth redirect in Supabase dashboard first",
    "Test with ngrok tunnel for local OAuth callback"
  ]
}
```

PHASE 7: HANDOFF - TRUE METHOD

Manual Iteration Process

NO PLUGIN AUTOMATION. Each user story requires:

1. **Open fresh Claude Code session**
2. **Execute single user story**
3. **Validate results**
4. **Close session**
5. **Open NEW session for next story**

Generate Execution Instructions

Provide the human with:

```
## READY FOR EXECUTION (TRUE METHOD)

### Execution Pattern:
For each user story, follow this cycle:

1. Open NEW Claude Code session (fresh context window)
2. Paste user story + PRD reference
3. Execute story to completion
4. Validate acceptance criteria met
5. Close session
6. Repeat for next story

### Pre-Flight Checklist:
- [ ] API keys in environment
- [ ] Dependencies installed
- [ ] Database migrations run (if applicable)
```

- [ ] Dev server accessible
- [ ] Git repo initialized (for iteration tracking)

Execution Order:
Stories are dependency-ordered. Execute in sequence:
- US-001: [title]
- US-002: [title]
- US-003: [title]
...

Expected Outcome:
- Each story completes in ONE fresh session (5-15 min)
- All acceptance criteria validated before next story
- Working code after each iteration
- No context accumulation between stories

Estimated:
~[X] sessions × [Y] min each = ~[Z] total time
(Each session is FRESH - optimal performance throughout)

Integration with Genesis:
After each successful story, update HANDOFF.md with:
- Story ID completed
- Files created/modified
- Next story to execute
- Any learnings/patterns discovered

ANTI-PATTERNS TO AVOID

❌ Don't Re-Ask What Was Already Said

BAD:

> *"What framework do you want to use?"*
> *(They already said "Next.js 14")*

GOOD:

> *"You mentioned Next.js 14 - should I use the app router or pages router?"*

❌ Don't Ask Open-Ended When Constrained

BAD:

> *"How should the UI look?"*

GOOD:

> *"For the dashboard layout, sidebar navigation or top nav bar?"*

❌ Don't Question Decisions, Clarify Implementation

BAD:

> *"Are you sure Supabase is the right choice?"*

GOOD:

> *"For Supabase auth, Row Level Security enabled or public access?"*

❌ Don't Overwhelm With Technical Jargon

BAD:

> *"What's your preferred state management paradigm for ephemeral vs persistent hydration?"*

GOOD:

> *"Should user settings persist on refresh (database) or reset (memory only)?"*

❌ Don't Create Multi-Session Stories

BAD:

```
{
  "title": "Build entire user management system",
  "acceptanceCriteria": ["CRUD operations, roles, permissions, audit logging"]
}
```

This will fail - context rot will degrade performance

GOOD:

```
{
  "title": "Create user CRUD endpoints",
  "acceptanceCriteria": ["POST /api/users creates user with validation"]
},
{
  "title": "Add role-based access control middleware",
  "acceptanceCriteria": ["Middleware blocks unauthorized routes based on role"]
}
```

Each completes in ONE fresh session

QUALITY GATES

Before finalizing PRD, verify:

✅ **Executability**: Agent can start work immediately with zero ambiguity
✅ **Atomicity**: Each user story completable in ONE fresh session (critical!)
✅ **Testability**: Clear pass/fail criteria for every requirement
✅ **Completeness**: No "TBD" or "figure it out later" placeholders
✅ **Bounded**: Clear scope - what's IN and what's OUT
✅ **Fresh-Session Compatible**: No stories requiring context persistence across sessions

---

SUCCESS METRICS

You've done this right when:

- Asked fewer than 15 clarifying questions total
- User said "yes exactly" to your readback
- Zero questions during agent execution about requirements
- First iteration produces runnable code in fresh session
- User doesn't come back saying "that's not what I meant"
- No story requires >15 min in fresh session (context rot indicator)

EXAMPLE INTERACTION FLOW

USER DICTATES: (10 min voice capture)

CLAUDE PARSES:

```
DECLARED: Next.js, Tailwind, Supabase, /dashboard path
IMPLIED: TypeScript (Next.js default), ESLint, Git
MISSING: API auth method, error handling specifics
RISKS: Multi-client switching logic underspecified
```

CLAUDE ASKS: (targeted questions)

> *Q1: "Stammer API auth method - Bearer token, API key header, or OAuth?"*
> *A1: "Bearer token, I have the key"*

> *Q2: "Multi-client switching - dropdown in header or separate client picker page?"*
> *A2: "Dropdown in header, top right"*

> *Q3: "Failed API calls - retry logic or show cached data?"*
> *A3: "Show cached data with warning banner"*

CLAUDE CONFIRMS: (structured readback)

```
PROJECT: AgileAdapt Client Dashboard
STACK: Next.js 14 (app router), TypeScript, Tailwind, Supabase
LOCATION: /app/dashboard
INTEGRATIONS: Stammer (Bearer), 10web (webhooks)
...
```

CLAUDE GENERATES: (agent-ready PRD)

```
{
  "project": {...},
  "userStories": [
    {
      "id": "US-001",
      "title": "Setup Next.js project with TypeScript and Tailwind",
      "acceptanceCriteria": [
        "GIVEN fresh project WHEN npx create-next-app runs THEN TypeScript and Tailwind
configured",
```

```
        "AND project runs on localhost:3000",
        "AND ESLint passes with no errors"
      ],
      "fileTargets": ["package.json", "tsconfig.json", "tailwind.config.ts"],
      "dependencies": []
    },
    {
      "id": "US-002",
      "title": "Setup Supabase client and auth helpers",
      ...
    }
  ]
}
```

CLAUDE PROVIDES: (execution instructions)

```
## Execute with TRUE Method:

Session 1: Open fresh Claude Code → Execute US-001 → Validate → Close
Session 2: Open fresh Claude Code → Execute US-002 → Validate → Close
Session 3: Open fresh Claude Code → Execute US-003 → Validate → Close
...

Each session gets MAXIMUM fresh context window.
No context rot. Optimal performance every iteration.
```

CONTINUOUS IMPROVEMENT

After each PRD:

1. Log questions that caused confusion → improve future templates
2. Track stories that needed human intervention → tighten acceptance criteria
3. Note gaps that block execution → add to mandatory checklist
4. Save successful dictation patterns → create few-shot examples
5. **Monitor story completion times → stories >15 min need atomic split**
6. **Update HANDOFF.md with execution patterns that work**

The goal: Each PRD better than the last. Each story atomic enough for ONE fresh session.

TRUE METHOD PHILOSOPHY

Old way (plugin): Automate iteration but accumulate context rot **TRUE method:** Manual iteration with fresh context every story

Trade-off: More human involvement per story **Benefit:** ZERO context degradation, optimal performance guaranteed

Genesis Integration: Each successful story updates HANDOFF.md, building system memory while maintaining execution freshness.