# MSW Protocol Architecture

## "Make Shit Work" - A Research-Grounded Autonomous Coding System

**Version:** 1.0
**Date:** February 2, 2026
**Status:** Architecture Specification

---

## Executive Summary

The MSW Protocol combines three proven patterns into a unified autonomous development system:

1. **GSD Protocol** - Spec-driven development with context engineering
2. **NotebookLM MCP** - Research grounding with zero-hallucination answers
3. **Ralph Wiggum Loop** - Continuous iteration until verifiable success

The result: You describe what you want, the system interviews you, researches best practices via NotebookLM, generates specs, and iterates autonomously until the code actually works.

---

## Problem Statement

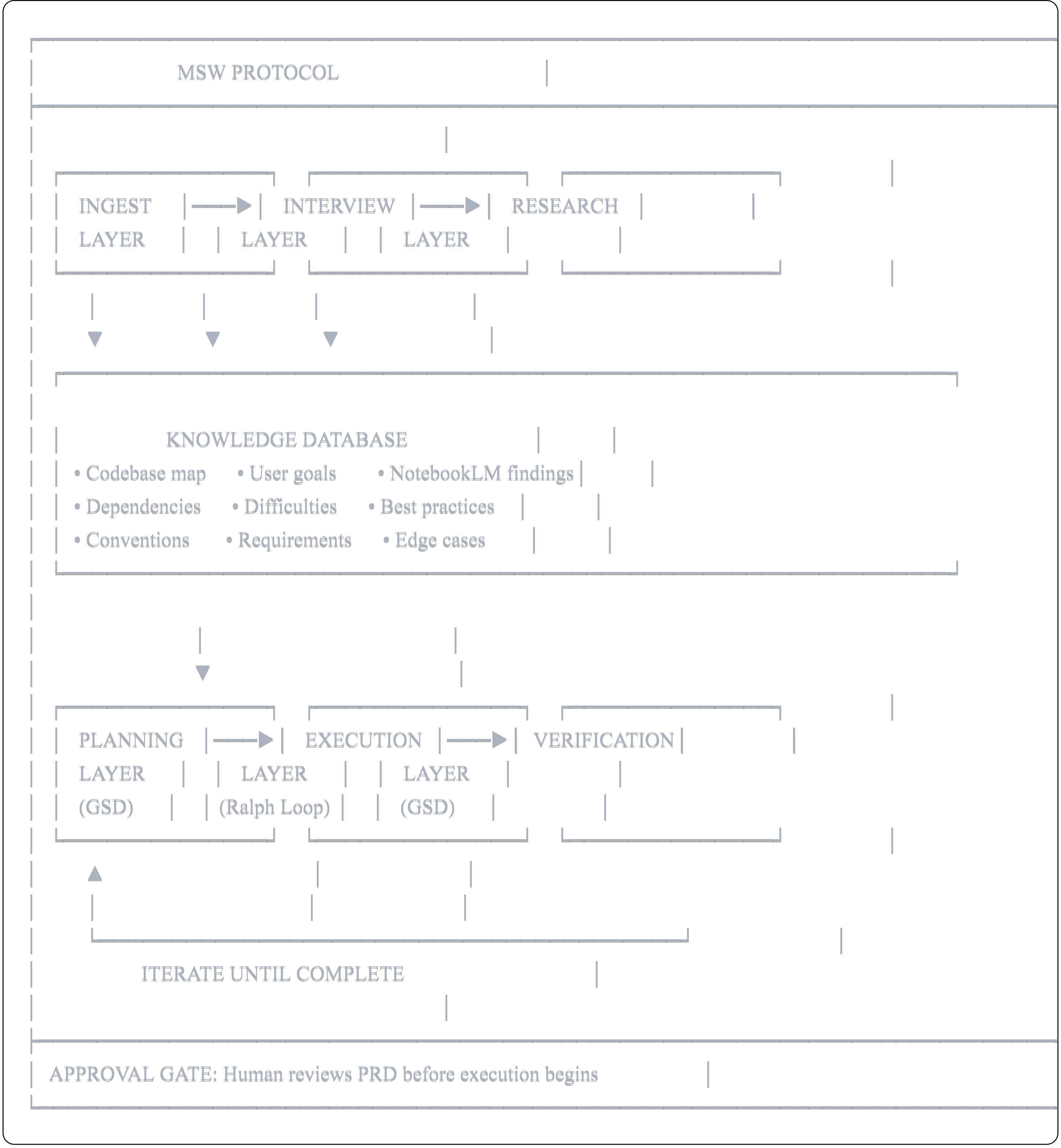
Current AI coding workflows suffer from:

| Problem | Symptom |
|---|---|
| **Context rot** | Quality degrades as context window fills |
| **Hallucinated APIs** | Agent invents methods that don't exist |
| **Premature exit** | Agent declares "done" before tests pass |
| **Knowledge gaps** | Agent lacks domain-specific best practices |
| **Manual research** | Copy-pasting between NotebookLM and IDE |
| **Endless loops** | No clear completion criteria |

MSW solves these by combining research grounding, spec-driven development, and autonomous iteration with verifiable success criteria.

---

## Architecture Overview



MSW PROTOCOL

INGEST LAYER ──▶ INTERVIEW LAYER ──▶ RESEARCH LAYER

KNOWLEDGE DATABASE

- Codebase map
- User goals
- NotebookLM findings
- Dependencies
- Difficulties
- Best practices
- Conventions
- Requirements
- Edge cases

PLANNING LAYER (GSD) ──▶ EXECUTION LAYER (Ralph Loop) ──▶ VERIFICATION LAYER (GSD)

ITERATE UNTIL COMPLETE

APPROVAL GATE: Human reviews PRD before execution begins

---

## Layer Specifications

### Layer 1: Ingest Layer

**Purpose:** Build a complete picture of the existing codebase

**Inputs:**

- Local directories (specified by user)
- GitHub repositories (via API or clone)
- Existing documentation

**Process:**

```
/msw:ingest --local ./src --github user/repo
```

**Outputs:**

```
.msw/
├── codebase/
│   ├── STRUCTURE.md      # Directory tree, file purposes
│   ├── DEPENDENCIES.md   # Package analysis, version info
│   ├── CONVENTIONS.md    # Code style, patterns detected
│   ├── ARCHITECTURE.md   # Component relationships
│   └── CONCERNS.md       # Tech debt, security issues
```

**Implementation:**

```
javascript

// Parallel agent orchestration (GSD pattern)
const ingestOrchestrator = {
  agents: [
    { name: 'structure-mapper', task: 'Analyze directory structure and file purposes' },
    { name: 'dependency-analyzer', task: 'Extract and analyze dependencies' },
    { name: 'pattern-detector', task: 'Identify code conventions and patterns' },
    { name: 'architecture-mapper', task: 'Map component relationships' },
    { name: 'concern-finder', task: 'Identify technical debt and issues' }
  ],
  async run(paths) {
    const results = await Promise.all(this.agents.map(a => spawnAgent(a)));
    return synthesize(results);
  }
};
```

---

**Layer 2: Interview Layer**

**Purpose:** Understand user's goals, difficulties, and requirements

**Process:**

```
/msw:interview
```

**Interview Flow:**

```
┌─────────────────────────────────────────────────────────────┐
│ INTERVIEW SESSION                                           │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│ Phase 1: Goals                                              │
│ ├─ "What are you trying to build/fix/improve?"              │
│ ├─ "What does success look like?"                           │
│ └─ "What's the deadline/priority?"                          │
│                                                             │
│ Phase 2: Difficulties                                       │
│ ├─ "What have you tried that didn't work?"                  │
│ ├─ "Where are you getting stuck?"                           │
│ └─ "What errors are you seeing?"                            │
│                                                             │
│ Phase 3: Agent Issues                                       │
│ ├─ "Paste any error messages from your coding agent"        │
│ ├─ "What did the agent attempt?"                            │
│ └─ "What was wrong with the agent's approach?"              │
│                                                             │
│ Phase 4: Constraints                                        │
│ ├─ "Any libraries/patterns you must/can't use?"             │
│ ├─ "Any existing code that must not change?"                │
│ └─ "Performance/security requirements?"                     │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

**Outputs:**

```
.msw/
├── interview/
│   ├── GOALS.md          # What user wants to achieve
│   ├── DIFFICULTIES.md   # What's been tried, what failed
│   ├── AGENT_ISSUES.md   # Errors from coding agents
│   └── CONSTRAINTS.md    # Hard requirements and limits
```

---

### Layer 3: Research Layer (NotebookLM Integration)

**Purpose:** Ground implementation decisions in authoritative documentation

**Key Innovation:** The system automatically formulates research questions based on the codebase analysis and interview, then queries NotebookLM for answers.

**Process:**

```
/msw:research
```

**Research Question Generation:**

```javascript
const questionFormulator = {
  async generateQuestions(context) {
    const { codebase, interview } = context;

    // Analyze gaps between what user wants and what codebase supports
    const gaps = analyzeGaps(codebase, interview);

    // Generate specific questions for NotebookLM
    return gaps.map(gap => ({
      topic: gap.area,
      question: `Given ${codebase.stack}, what is the recommended approach for ${gap.requirement}? Include specific API me
      notebook: selectBestNotebook(gap.area), // Auto-select from library
      followUps: generateFollowUps(gap)
    }));
  }
};
```

**NotebookLM MCP Integration:**

```javascript
// Using notebooklm-mcp
const notebookLM = {
  async research(questions) {
    const findings = [];

    for (const q of questions) {
      // Activate the relevant notebook
      await mcpClient.call('notebooklm_activate', {
        notebook_id: q.notebook
      });

      // Ask primary question
      const answer = await mcpClient.call('notebooklm_ask', {
        question: q.question
      });
      findings.push({ topic: q.topic, primary: answer });

      // Follow-up questions for depth
      for (const followUp of q.followUps) {
        const detail = await mcpClient.call('notebooklm_ask', {
          question: followUp
        });
        findings[findings.length - 1].details.push(detail);
      }
    }

    return findings;
  }
};
```

**Notebook Library Management:**

```yaml
# .msw/notebooks.yaml
notebooks:
  - id: react-docs
    url: https://notebooklm.google.com/notebook/abc123
    topics: [react, hooks, components, state]
    description: "Official React documentation + patterns"

  - id: nodejs-api
    url: https://notebooklm.google.com/notebook/def456
    topics: [node, express, api, backend]
    description: "Node.js API best practices"

  - id: project-specific
    url: https://notebooklm.google.com/notebook/ghi789
    topics: [custom, domain]
    description: "Our internal docs and patterns"
```

**Outputs:**

```
.msw/
├── research/
│   ├── QUESTIONS.md      # Generated research questions
│   ├── FINDINGS.md       # NotebookLM responses (grounded)
│   ├── BEST_PRACTICES.md # Synthesized recommendations
│   └── EDGE_CASES.md     # Pitfalls to avoid
```

---

### Layer 4: Planning Layer (GSD Integration)

**Purpose:** Generate implementation specifications grounded in research

**Process:**

```
/msw:plan
```

**PRD Generation:**

javascript

```javascript
const prdGenerator = {
  async generate(context) {
    const { codebase, interview, research } = context;

    return {
      // From interview
      goals: interview.goals,
      constraints: interview.constraints,

      // From codebase analysis
      existingPatterns: codebase.conventions,
      modificationScope: codebase.architecture,

      // From NotebookLM research
      implementation: research.bestPractices,
      avoidPitfalls: research.edgeCases,

      // GSD-style phases
      phases: generatePhases(interview.goals, research.findings),

      // Success criteria (for Ralph loop)
      completionCriteria: {
        tests: 'All tests pass with >80% coverage',
        lint: 'No linter errors',
        build: 'Build succeeds',
        integration: interview.specificCriteria
      }
    };
  }
};
```

**Outputs:**

```
.msw/
├── planning/
│   ├── PRD.md            # Full specification document
│   ├── ROADMAP.md        # Phase breakdown (GSD format)
│   ├── REQUIREMENTS.md   # Traceable requirements
│   └── phases/
│       ├── 01-PLAN.md    # Task-level XML plans
│       ├── 02-PLAN.md
│       └── ...
```

**Plan Format (GSD XML):**

```xml
<task type="auto">
  <n>Implement user authentication</n>
  <files>src/auth/login.ts, src/auth/middleware.ts</files>
  <action>
    Based on NotebookLM research (FINDINGS.md#jwt-best-practices):
    - Use jose library for JWT (not jsonwebtoken - CommonJS issues)
    - Implement refresh token rotation
    - Store tokens in httpOnly cookies

    Existing pattern to follow (CONVENTIONS.md#error-handling):
    - Use AppError class for all errors
    - Return standardized error responses
  </action>
  <verify>
    - curl -X POST /auth/login returns 200 + Set-Cookie
    - Invalid credentials return 401 with error body
    - Token refresh works before expiry
  </verify>
  <done>Auth flow complete with tests passing</done>
</task>
```

---

### Layer 5: Approval Gate

**Purpose:** Human review before autonomous execution begins

**Process:**

```
/msw:review
```

**Review Interface:**

```
┌─────────────────────────────────────────────────────────────┐
│ MSW APPROVAL GATE                                           │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│ PRD Summary:                                                │
│ ┌─────────────────────────────────────────────────────────┐ │
│ │ Goal: Add JWT authentication to API                     │ │
│ │ Phases: 4                                               │ │
│ │ Estimated tasks: 12                                     │ │
│ │ Research sources: 3 NotebookLM notebooks                │ │
│ └─────────────────────────────────────────────────────────┘ │
│                                                             │
│ Key Decisions (from NotebookLM research):                   │
│ • Using jose over jsonwebtoken (CommonJS compatibility)     │
│ • Refresh token rotation enabled                            │
│ • Storing in httpOnly cookies (not localStorage)            │
│                                                             │
│ Files to modify: 8                                          │
│ New files to create: 4                                      │
│                                                             │
│ [View Full PRD] [View Research] [View Plans]                │
│                                                             │
│ ┌──────────┐ ┌──────────┐ ┌──────────┐                      │
│ │ APPROVE  │ │ EDIT     │ │ REJECT   │                      │
│ └──────────┘ └──────────┘ └──────────┘                      │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

**Approval Actions:**

- **APPROVE** → Proceed to execution
- **EDIT** → Return to planning with feedback
- **REJECT** → Cancel and explain why

---

### Layer 6: Execution Layer (Ralph Wiggum Loop)

**Purpose:** Autonomous implementation with continuous iteration

**Process:**

```
/msw:execute --phase 1 --max-iterations 30
```

**Ralph Loop Integration:**

javascript

```javascript
const mswExecutor = {
  async executePhase(phaseNum, maxIterations = 30) {
    const plan = loadPlan(phaseNum);
    const completionPromise = plan.completionCriteria;

    // Ralph loop wrapper
    let iteration = 0;
    while (iteration < maxIterations) {
      iteration++;

      // Execute with fresh context (GSD pattern)
      const result = await spawnExecutorAgent({
        plan: plan,
        research: loadResearch(), // NotebookLM findings available
        iteration: iteration,
        previousAttempts: loadSummaries(phaseNum)
      });

      // Check completion (Ralph pattern)
      if (await verifyCompletion(completionPromise)) {
        await commitWork(result);
        return { success: true, iterations: iteration };
      }

      // Query NotebookLM if stuck
      if (result.stuck) {
        const help = await queryNotebookLM(result.stuckReason);
        await updateContext(help);
      }

      // Log iteration
      await logIteration(iteration, result);
    }

    return { success: false, reason: 'Max iterations reached' };
  }
};
```

### Feedback Loop with NotebookLM:

```javascript
const feedbackLoop = {
  async onStuck(context) {
    const { error, attemptedApproach, codeFile } = context;

    // Formulate question from error
    const question = `
      I'm implementing ${context.task} and getting this error: ${error}

      My approach: ${attemptedApproach}

      What's the correct way to handle this? Provide specific code patterns.
    `;

    // Query relevant notebook
    const answer = await mcpClient.call('notebooklm_ask', {
      question,
      notebook_id: selectNotebookForError(error)
    });

    // Feed answer back to executor
    return {
      guidance: answer,
      retry: true
    };
  }
};
```

**Outputs:**

```
.msw/
├── execution/
│   ├── phase-01/
│   │   ├── iteration-001.log
│   │   ├── iteration-002.log
│   │   ├── ...
│   │   └── SUMMARY.md
│   └── phase-02/
│       └── ...
```

---

**Layer 7: Verification Layer**

**Purpose:** Confirm implementation actually works

**Process:**

```
/msw:verify --phase 1
```

**Verification Checks:**

```javascript
const verifier = {
  checks: [
    // Automated
    { name: 'tests', command: 'npm test', expect: 'exit 0' },
    { name: 'lint', command: 'npm run lint', expect: 'exit 0' },
    { name: 'build', command: 'npm run build', expect: 'exit 0' },
    { name: 'coverage', command: 'npm run coverage', expect: '>80%' },

    // From plan verification steps
    { name: 'api-login', command: 'curl -X POST /auth/login', expect: '200' },
    { name: 'api-protected', command: 'curl /protected', expect: '401' }
  ],

  async verify(phase) {
    const results = [];

    for (const check of this.checks) {
      const result = await runCheck(check);
      results.push({ check: check.name, passed: result.passed, output: result.output });

      if (!result.passed) {
        // Query NotebookLM for fix guidance
        const guidance = await queryNotebookLM(
          `How to fix: ${check.name} failed with ${result.output}`
        );
        results[results.length - 1].guidance = guidance;
      }
    }

    return results;
  }
};
```

**If Verification Fails:**

1. System generates fix plans with NotebookLM guidance
2. Returns to Execution Layer
3. Ralph loop continues until all checks pass

---

## Complete Workflow

```
┌──────────────────────────────────────────────────────────────────
│                    MSW PROTOCOL WORKFLOW                         │
└──────────────────────────────────────────────────────────────────

User: /msw:init --local ./src --github user/repo

   ┌──────────────────────┐
   │ 1. INGEST        │ Analyze codebase structure, dependencies, patterns
   └──────────┬───────────┘
       │
       ▼
   ┌──────────────────────┐
   │ 2. INTERVIEW    │ Ask about goals, difficulties, agent issues
   └──────────┬───────────┘
       │
       ▼
   ┌──────────────────────┐
   │ 3. RESEARCH     │ Query NotebookLM with generated questions
   └──────────┬───────────┘
       │
       ▼
   ┌──────────────────────┐
   │ 4. PLAN        │ Generate PRD and phase plans (GSD format)
   └──────────┬───────────┘
       │
       ▼
   ┌──────────────────────┐
   │ 5. APPROVAL     │ Human reviews before execution
   └──────────┬───────────┘
       │
       ▼
   ┌──────────────────────┐
   │ 6. EXECUTE      │ Ralph loop per phase with NotebookLM feedback
   │    (LOOP)      │
   └──────────┬───────────┘
       │
       ▼
   ┌──────────────────────┐
   │ 7. VERIFY        │ Confirm everything works
   └──────────┬───────────┘
       │
       ▼
    ┌───────────┐
    │ DONE?  │──NO──▶ Back to EXECUTE (next phase or fix)
    └─────┬─────┘
      │
      |
     YES
      |
      ▼
   ┌───────────────┐
   │ COMPLETE      │ Tag release, archive milestone
   └───────────────┘
```

## Implementation Options

### Option A: MCP Server (Recommended)

Create a unified MCP server that exposes all MSW commands:

javascript

```javascript
// msw-mcp-server/index.js
import { Server } from '@modelcontextprotocol/sdk/server';
import { NotebookLMMCP } from 'notebooklm-mcp';

const server = new Server({
  name: 'msw-protocol',
  version: '1.0.0'
});

// Tools
server.addTool('msw_init', 'Initialize MSW for a project', async (params) => {
  await ingestLayer.run(params.local, params.github);
  await interviewLayer.run();
  return { status: 'ready', next: 'msw_research' };
});

server.addTool('msw_research', 'Research via NotebookLM', async () => {
  const questions = await questionFormulator.generateQuestions(context);
  const findings = await notebookLM.research(questions);
  return { findings, next: 'msw_plan' };
});

server.addTool('msw_plan', 'Generate PRD and plans', async () => {
  const prd = await prdGenerator.generate(context);
  return { prd, awaitingApproval: true };
});

server.addTool('msw_execute', 'Execute with Ralph loop', async (params) => {
  return await mswExecutor.executePhase(params.phase, params.maxIterations);
});

server.addTool('msw_verify', 'Verify implementation', async (params) => {
  return await verifier.verify(params.phase);
});
```

### Windsurf Integration:

```
json

// ~/.codeium/windsurf/mcp_config.json
{
 "mcpServers": {
  "msw": {
   "command": "node",
   "args": ["/path/to/msw-mcp-server/index.js"]
  },
  "notebooklm": {
   "command": "npx",
   "args": ["-y", "notebooklm-mcp@latest"]
  }
 }
}
```

### Option B: Claude Code Skill

Create as a Claude Code skill for direct integration:

```
~/.claude/skills/msw/
├── SKILL.md
├── commands/
│   ├── init.md
│   ├── interview.md
│   ├── research.md
│   ├── plan.md
│   ├── execute.md
│   └── verify.md
└── scripts/
    ├── ingest.py
    ├── research.py
    └── execute.py
```

### Option C: GSD Extension

Fork and extend the GSD protocol:

```javascript
// Extend GSD with NotebookLM integration
import { GSD } from 'get-shit-done-cc';
import { NotebookLMMCP } from 'notebooklm-mcp';

class MSW extends GSD {
  async planPhase(phaseNum) {
    // Standard GSD planning + NotebookLM research
    const questions = this.generateResearchQuestions(phaseNum);
    const research = await this.notebookLM.research(questions);

    // Inject research into plan context
    this.context.research = research;

    return super.planPhase(phaseNum);
  }

  async executePhase(phaseNum) {
    // Standard GSD execution + Ralph loop
    return this.ralphLoop(
      () => super.executePhase(phaseNum),
      this.getCompletionCriteria(phaseNum)
    );
  }
}
```

---

## Directory Structure

```
project/
├── .msw/
│   ├── config.yaml           # MSW configuration
│   ├── notebooks.yaml        # NotebookLM library
│   │
│   ├── codebase/             # From ingest layer
│   │   ├── STRUCTURE.md
│   │   ├── DEPENDENCIES.md
│   │   ├── CONVENTIONS.md
│   │   ├── ARCHITECTURE.md
│   │   └── CONCERNS.md
│   │
│   ├── interview/            # From interview layer
│   │   ├── GOALS.md
│   │   ├── DIFFICULTIES.md
│   │   ├── AGENT_ISSUES.md
│   │   └── CONSTRAINTS.md
│   │
│   ├── research/             # From research layer
│   │   ├── QUESTIONS.md
│   │   ├── FINDINGS.md
│   │   ├── BEST_PRACTICES.md
│   │   └── EDGE_CASES.md
│   │
│   ├── planning/             # From planning layer
│   │   ├── PRD.md
│   │   ├── ROADMAP.md
│   │   ├── REQUIREMENTS.md
│   │   └── phases/
│   │       ├── 01-PLAN.md
│   │       └── ...
│   │
│   └── execution/            # From execution layer
│       ├── phase-01/
│       │   ├── iteration-001.log
│       │   └── SUMMARY.md
│       └── ...
│
└── .planning/                # GSD compatibility (optional)
    └── ...
```

---

## Configuration

yaml

```yaml
# .msw/config.yaml
msw:
  version: "1.0"

  # Ingest settings
  ingest:
    local_paths:
      - ./src
      - ./lib
    github_repos:
      - user/main-repo
      - user/shared-lib
    exclude:
      - node_modules
      - dist
      - .git

  # NotebookLM settings
  notebooklm:
    default_notebooks:
      - react-docs
      - nodejs-api
    auto_select: true          # Automatically pick notebook by topic
    max_questions_per_topic: 5
    follow_up_depth: 3

  # Execution settings
  execution:
    model_profile: balanced    # quality | balanced | budget
    max_iterations: 30         # Ralph loop limit
    parallel_execution: true   # Run independent tasks in parallel

  # Verification settings
  verification:
    auto_tests: true
    coverage_threshold: 80
    custom_checks:
      - name: api-health
        command: curl -f http://localhost:3000/health

  # Approval settings
  approval:
    mode: interactive          # interactive | telegram | slack
    telegram_chat_id: "..."    # If using telegram
```

## Commands Reference

| Command | Description |
|---|---|
| /msw:init | Full initialization: ingest → interview → research → plan |
| /msw:ingest | Analyze codebase only |
| /msw:interview | Run interview session only |
| /msw:research | Query NotebookLM based on current context |
| /msw:plan | Generate PRD and phase plans |
| /msw:review | Human approval gate |
| /msw:execute [phase] | Execute with Ralph loop |
| /msw:verify [phase] | Verify implementation |
| /msw:status | Show current progress |
| /msw:notebook add <url> | Add NotebookLM to library |
| /msw:notebook list | List available notebooks |
| /msw:config | Edit configuration |

## Integration with Existing Tools

### With GSD Protocol

MSW can wrap GSD completely:

```
/msw:init        # Does ingest + interview + research + /gsd:new-project
/msw:execute 1    # Does /gsd:plan-phase 1 + Ralph loop on /gsd:execute-phase 1
```

### With Ralph Wiggum Plugin

MSW uses Ralph internally:

javascript

```javascript
// MSW calls Ralph for execution
const result = await ralphLoop({
  prompt: plan.toPrompt(),
  completionPromise: plan.completionCriteria,
  maxIterations: config.execution.maxIterations,
  onIteration: (i, result) => {
    if (result.stuck) {
      // MSW-specific: query NotebookLM
      return queryNotebookLM(result.stuckReason);
    }
  }
});
```

### With NotebookLM MCP

MSW orchestrates NotebookLM queries:

javascript

```javascript
// MSW manages the NotebookLM connection
const mcpClient = await connect('notebooklm');

// Auto-select notebook based on topic
const notebook = await selectBestNotebook(topic, library);
await mcpClient.call('notebooklm_activate', { notebook_id: notebook.id });

// Ask with follow-ups
const answer = await mcpClient.call('notebooklm_ask', { question });
```

---

## Success Criteria

MSW is considered successful when:

1. **Zero copy-paste** - No manual transfer between NotebookLM and IDE
2. **Grounded implementations** - All major decisions backed by NotebookLM research
3. **Verifiable completion** - Tests pass, build succeeds, custom checks pass
4. **Autonomous iteration** - Ralph loop handles retries without human intervention
5. **Clear audit trail** - Every decision, question, and iteration logged

---

## Next Steps

1. **Prototype MCP Server** - Build minimal MSW MCP with core workflow
2. **NotebookLM Integration** - Test question formulation and auto-selection
3. **GSD Extension** - Fork GSD and add research layer
4. **Ralph Integration** - Wire up execution layer with completion criteria
5. **Approval Gate UI** - Build Telegram/CLI interface for reviews
6. **Open Source** - Release on GitHub with MIT license

---

## Appendix: Example Session

```
$ /msw:init --local ./src --github myuser/myapp

MSW Protocol v1.0 - Initializing...

[INGEST] Analyzing codebase...
  ✓ Structure mapped (47 files, 12 directories)
  ✓ Dependencies analyzed (23 packages)
  ✓ Conventions detected (ESLint + Prettier, React patterns)
  ✓ Architecture mapped (3 layers: API, Services, Components)
  ✓ Concerns found (2 security issues, 5 tech debt items)

[INTERVIEW] Starting interview session...

  Q: What are you trying to build?
  > Add user authentication with JWT tokens

  Q: What have you tried that didn't work?
  > Claude kept using jsonwebtoken which has CommonJS issues

  Q: Paste any error messages:
  > [ERR_REQUIRE_ESM]: require() of ES Module not supported

  Q: Any libraries you must use?
  > Need to support refresh tokens and httpOnly cookies

[RESEARCH] Querying NotebookLM...
  ✓ Selected notebook: nodejs-api (topic match: 94%)
  ✓ Asked 5 questions with 12 follow-ups
  ✓ Best practice: Use jose library (ESM compatible)
  ✓ Best practice: Implement refresh token rotation
  ✓ Edge case: Handle token refresh race conditions

[PLAN] Generating PRD...
  ✓ Created PRD.md
  ✓ Created ROADMAP.md (4 phases)
  ✓ Created 12 task plans

[APPROVAL] Awaiting review...
  PRD: Authentication System with JWT
  Phases: 4
  Tasks: 12
  Research sources: 3

  [APPROVE] [EDIT] [REJECT]
  > APPROVE
```

```
[EXECUTE] Starting Phase 1 with Ralph loop...
  Iteration 1: Created auth middleware... tests failing
  Iteration 2: Fixed token validation... 2 tests failing
  Iteration 3: Added refresh logic... all tests passing
  ✓ Phase 1 complete in 3 iterations

[VERIFY] Running verification...
  ✓ npm test - 24/24 passed
  ✓ npm run lint - 0 errors
  ✓ npm run build - success
  ✓ Coverage: 87%
  ✓ POST /auth/login - 200 OK

MSW Complete. Authentication system implemented and verified.
```

---

*"No more endless loops. No more fuckery on coding agents. Just Make Shit Work."*