Product Requirements Document (PRD)

CoreSkills4ai Autonomous Development System

Version: 1.0.0
Last Updated: 2026-01-11
Status: Foundation Phase
Owner: GameRoom PC Shaw1

🎯 Mission Statement

Build a Markdown-driven autonomous development system that executes production-grade workflows across multiple development modes (scoping → security → testing → fixing), self-validates continuously, and generates portfolio-worthy projects while teaching engineering best practices.

📋 Executive Summary

What We're Building

An AI agent system that:

1. **Reads Markdown instructions** to understand tasks, constraints, and context
2. **Operates autonomously** with minimal daily prompting
3. **Switches operational modes** based on development phase (scoping, coding, testing, fixing)
4. **Self-validates** work against explicit rules
5. **Teaches** engineering patterns through generated classroom materials
6. **Produces** portfolio-quality projects for career advancement

Who It's For

- **Primary User:** Solo developer building AI engineering portfolio
- **Secondary Users:** Future students in classroom environments
- **Tertiary Users:** Potential employers/clients viewing portfolio work

Success Criteria

- ✅ Ship 3+ portfolio-worthy projects in 6 months
- ✅ Reduce daily coding intervention by 70%
- ✅ Generate classroom-ready materials automatically
- ✅ Land AI/Systems Engineering role

🏗️ System Architecture Overview

Core Components

1. Agent Runtime

- **Language:** Rust (performance) + Python (tooling)
- **Responsibility:** Execute tasks from Markdown instructions
- **Key Features:**
  - Parse structured Markdown (PRD, TASKS, RULES)
  - Execute workflows autonomously
  - Handle errors with escalation
  - Self-update rules based on learnings

2. Mode System (Multi-Phase Operation)

Different operational "vibes" for different development phases:

Mode	Purpose	Rules Tightness	Autonomy Level
SCOPE	Define requirements, plan architecture	Loose - ask many questions	Low - collaborative
BUILD	Fast feature implementation	Medium - ship quickly	High - minimal interruption
SECURE	Add auth, validation, hardening	Tight - no shortcuts	Medium - verify everything
TEST	Write tests, find bugs	Tight - comprehensive	High - automated validation
FIX	Debug, patch, optimize	Medium - targeted fixes	Medium - suggest solutions
DEPLOY	Package, containerize, ship	Tight - production-ready	Low - manual approval

Implementation: Separate Markdown files per mode:

- `modes/SCOPE.md` - Scoping rules & templates
- `modes/BUILD.md` - Build phase rules
- `modes/SECURE.md` - Security requirements
- etc.

3. Context Engine

- **RAG (Retrieval-Augmented Generation):** Index all Markdown, code, docs
- **MCP Integration:** Filesystem, Docker, future Git integration
- **Memory System:** Persist architecture decisions, mistakes, preferences

4. Task Orchestrator

- **Input:** `TASKS.md` with structured task list
- **Processing:** Parse, prioritize, execute, validate
- **Output:** Updated task status, end-of-day report

5. Docker Automation

- **Dev Containers:** Python 3.12, Node.js, Rust environments
- **Classroom Containers:** Pre-built lesson environments
- **Production Images:** Portfolio project deployments

6. Educational Layer

- **Auto-Documentation:** Generate tutorials from code
- **Template Library:** Reusable project scaffolds
- **Portfolio Generator:** Showcase projects professionally

🎯 Use Cases

UC1: Daily Development Session

Actor: Developer
Flow:

1. Developer opens `TASKS.md`, sees today's priority
2. Agent reads task, enters appropriate MODE
3. Agent executes work autonomously
4. Agent updates `TASKS.md` with progress
5. Agent generates end-of-day summary
6. Developer reviews, approves, commits

UC2: Mode Switching (Scope → Build)

Actor: Agent System
Flow:

1. Complete scoping phase (PRD finalized)
2. Developer switches: "Enter BUILD mode"
3. Agent loads `modes/BUILD.md` rules
4. Agent autonomy increases, question frequency decreases
5. Agent ships features rapidly with minimal prompting

UC3: Portfolio Project Creation

Actor: Developer
Flow:

1. Developer: "Create portfolio project: AI chatbot"
2. Agent enters SCOPE mode, asks clarifying questions
3. Agent generates `PRD.md`, `ARCHITECTURE.md`, `TASKS.md`
4. Developer approves
5. Agent switches to BUILD mode, implements features
6. Agent generates classroom materials automatically

7. Agent containerizes for deployment

UC4: Error Recovery & Learning

Actor: Agent System
Flow:

1. Agent encounters error during BUILD
2. Agent logs error, switches to FIX mode
3. Agent suggests solutions, developer approves
4. Agent implements fix
5. Agent updates `DECISIONS.md` with lesson learned
6. Agent updates `modes/BUILD.md` to prevent future occurrence

🔗 Technical Requirements

Hard Constraints (Non-Negotiable)

Hardware

- **Current:** Weak local machine (i5-7600K, 16GB RAM, GTX 1070)
- **Optimization:** Minimize local compute, offload to cloud when possible
- **GPU:** Not expected for most tasks

Operating System

- **Primary:** Windows 10 Home
- **WSL2:** Required for Docker, Linux tooling
- **Tooling:** Must work in both Windows and WSL2

Python

- **Version:** 3.12.x (stability over bleeding-edge)
- **Role:** Primary language for agent runtime, APIs, tooling
- **Key Libraries:**
  - `numpy`, `pandas` (data)

- `chromadb`, `sentence-transformers` (RAG)
- `ollama` (local LLM)
- `streamlit` (UI)
- `pyyaml`, `python-dotenv` (config)

Node.js

- **Role:** Frontend, CLI tools, MCP integration
- **Version:** Latest LTS (check with `node --version`)

Rust

- **Role:** Performance-critical paths (optional, not required initially)
- **FFI:** Plan boundaries later if needed

Docker

- **Philosophy:** One container per concern (modular, reusable)
- **Strategy:** Dev containers now, production images later
- **Agent Capabilities:**
  - Build images (with approval for critical changes)
  - Run containers
  - Modify Dockerfiles (prompt for critical changes)

Security Posture

- **Untrusted Code:** Never execute without approval
- **Secrets:** Store in MCP servers or environment variables
- **User Data:** Protect via MCP integration, sandbox environments

Offline Tolerance

- **Internet Required:** System expects API access
- **Occasional Offline:** Laptop work sometimes offline-capable
- **API Access:** Not guaranteed 100% uptime

📊 Data & State Management

Truth Sources

- **Database:** Persistent application state
- **Markdown:** Architecture decisions, tasks, rules
- **JSON:** Configuration artifacts
- **Git:** Version control (not source of truth for agent state)

Configuration Formats

- **YAML:** Primary config format
- **TOML:** Alternative for specific tools
- **Markdown:** Human-readable rules/docs
- **Environment Variables:** Runtime secrets

Agent Behavior Rules

Autonomy Levels (By Mode)

- **SCOPE Mode:** Low autonomy - ask many questions
- **BUILD Mode:** High autonomy - ship fast, minimal interruptions
- **SECURE Mode:** Medium autonomy - verify everything
- **TEST Mode:** High autonomy - automated validation
- **FIX Mode:** Medium autonomy - suggest, then implement
- **DEPLOY Mode:** Low autonomy - manual approval required

Question Policy

When to Ask:

- ✅ Missing requirements (always)
- ✅ Any ambiguity (always)

- ✅ Critical file modifications (system-impacting changes)
- ✅ New dependencies (prompt first)
- ✅ Structural refactoring (explain risks)

When NOT to Ask:

- ❌ During heavy BUILD mode (assume sensible defaults)
- ❌ For trivial code style choices
- ❌ For obvious fixes during FIX mode

Repetition Rule

"Anything sent more than twice becomes a command"

- **Auto-Codify:** Convert repeated instructions into rules automatically
- **Surface for Confirmation:** Agent proposes new rules for approval
- **Store in Mode Files:** Add to appropriate `modes/*.md`

Error Handling

- **Fail Fast:** Stop early if approach seems wrong
- **Self-Correct Silently:** Fix minor issues without escalation
- **Escalate Critical Errors:** Always notify with suggested solutions
- **Learn & Update:** Add learnings to `DECISIONS.md`, update mode rules

File Creation Policy

- **Safe to Create Without Approval:**
  - Documentation files (`.md`, `.txt`)
  - Config templates (`.yaml`, `.json` in `templates/`)
  - Test files
  - Logs/reports
- **Require Approval:**
  - Core system files (`ARCHITECTURE.md`, mode definitions)
  - Dockerfile modifications
  - Database migrations

- Security-related changes

📝 Mandatory Artifacts

Core Documents (Agent Maintains)

1. **PRD.md** (this file) - Product requirements
2. **ARCHITECTURE.md** - System design, component details
3. **TASKS.md** - Task queue, status tracking
4. **DECISIONS.md** - Architecture decisions, learnings
5. **RUNBOOK.md** - Operational procedures, deployment

Mode Definitions

- `modes/SCOPE.md` - Scoping phase rules
- `modes/BUILD.md` - Build phase rules
- `modes/SECURE.md` - Security phase rules
- `modes/TEST.md` - Testing phase rules
- `modes/FIX.md` - Debug phase rules
- `modes/DEPLOY.md` - Deployment phase rules

Update Authority

- **Autonomous Updates:** TASKS.md, daily reports, logs
- **Approval Required:** PRD.md, ARCHITECTURE.md, mode definitions
- **Propose Diffs:** Agent suggests changes, developer approves

Markdown Standards

- **Enforced Templates:** Yes (structured sections required)
- **Professional Quality:** Clean, organized, classroom-ready
- **Version Controlled:** All changes committed to Git

🎓 Educational Integration

Classroom Materials (Auto-Generated)

- **Tutorial Docs:** Generated from implemented features
- **Code Walkthroughs:** Annotated explanations
- **Lesson Plans:** Step-by-step teaching guides
- **Exercise Sets:** Practice problems based on project code

Portfolio Outputs

- **Project Showcases:** Professional README, screenshots
- **Case Studies:** Problem → Solution documentation
- **Technical Write-Ups:** Architecture explanations
- **Demo Videos:** (future - scripted by agent)

🚀 Deployment & CI/CD

GitHub Actions

- **Automated Testing:** Run tests on push (future)
- **Container Builds:** Auto-build Docker images
- **Documentation:** Deploy docs to GitHub Pages

Deployment Targets

- **Local:** Docker containers for development
- **Cloud:** VM/container hosting (future)
- **Distribution:** Open-source when appropriate (per-project decision)

📈 Success Metrics

Quantitative

- **Portfolio Projects:** 3+ shipped in 6 months
- **Daily Intervention:** <30% of development time
- **Code Quality:** Linted, documented, containerized
- **Classroom Materials:** 1 lesson per project feature

Qualitative

- **Portfolio Impression:** "Wow, this person knows production systems"
- **Agent Reliability:** "I trust it to build while I sleep"
- **Learning Outcomes:** "I understand the patterns it taught me"

🚧 Roadmap

Phase 1: Foundation (Week 1) ← WE ARE HERE

- ✅ Repository structure
- ✅ Core Markdown documents (PRD, ARCHITECTURE, TASKS)
- ✅ Docker dev environment
- ✅ Mode system design

Phase 2: Agent Core (Week 2-3)

- [ ] Markdown parser (read PRD, TASKS, RULES)
- [ ] Task executor (basic workflow automation)
- [ ] Error handling & escalation
- [ ] Mode switching logic

Phase 3: Context Engine (Week 4-5)

- [ ] RAG indexing (code, docs, Markdown)
- [ ] MCP integration (filesystem, Docker)
- [ ] Memory persistence (decisions, learnings)

Phase 4: First Portfolio Project (Week 6-8)

- [ ] SCOPE → BUILD → DEPLOY full cycle
- [ ] Auto-generated classroom materials
- [ ] Professional showcase deployment

Phase 5: Iteration & Jobs (Week 9-12)

- [ ] Second portfolio project
- [ ] Resume updates with project links
- [ ] Job applications (AI/Systems Engineer roles)

🔒 Constraints & Limitations

What This System WILL NOT Do

- ❌ Execute untrusted code without approval
- ❌ Overwrite files without version control safety
- ❌ Make architectural decisions without documented rationale
- ❌ Deploy to production without explicit approval
- ❌ Share secrets or sensitive data insecurely

Known Limitations

- **Local Compute:** Limited GPU, may need cloud for heavy ML
- **Internet Dependency:** Requires API access for full functionality
- **Manual Approval:** Critical changes require human review
- **Learning Curve:** Agent improves over time, early iterations need guidance

---

📞 Support & Escalation

When Agent Is Stuck

1. Log issue in `TASKS.md` with BLOCKED status
2. Generate error report with context

3. Suggest 3 possible solutions
4. Wait for developer decision

When Developer Is Stuck

1. Review `DECISIONS.md` for past solutions
2. Check mode-specific rules in `modes/*.md`
3. Consult this PRD for system constraints
4. Ask agent to explain its reasoning

📚 Appendix

Glossary

- **Mode:** Operational phase with specific rules (SCOPE, BUILD, etc.)
- **RAG:** Retrieval-Augmented Generation (context from docs)
- **MCP:** Model Context Protocol (tool integration)
- **Artifact:** Generated output (code, docs, configs)
- **PRD:** Product Requirements Document (this file)

References

- Agent System Repo: `C:\CoreSkills4ai\agent-system\`
- Projects Folder: `C:\CoreSkills4ai\projects\`
- Shared Templates: `C:\CoreSkills4ai\shared\templates\`

END OF PRD v1.0.0

Next Update: After Phase 1 completion (expected: 2026-01-18)