AI-Driven Solo Development: A Process-Oriented Framework

From Ideation to Deployment Using Agents.md and YAML Governance

Executive Summary

This document outlines a structured, AI-driven development framework designed specifically for solo developers ("watchers") who want to build scalable software from concept to production. The framework leverages agents.md cascading configuration and YAML-driven workflows to transform novice builders into capable developers through four distinct phases: **IDEATION**, **Proof of Concept (POC)**, **Minimum Viable Product (MVP)**, and **DEPLOYMENT**.

Each phase includes specific prompts, tools, and governance patterns that work synergistically with AI coding assistants like Cursor, GitHub Copilot, and Claude to create a repeatable, learnable development process.

Phase 1: IDEATION - Concept Discovery and Validation

Objective

Transform vague ideas into structured, validated concepts ready for technical implementation.

Key Activities

- Problem identification and market research
- User persona development
- Feature prioritization
- Technical feasibility assessment
- Success metrics definition

AI Prompts for Ideation Phase

1. Problem Definition Prompt

```
You are an experienced startup advisor. I have an idea for [BRIEF_DESCRIPTION].

Please help me:
1. Clearly define the core problem this solves
2. Identify 3-5 specific user pain points
3. Suggest 3 alternative approaches to solving this problem
4. Rate the market opportunity (1-10) with reasoning
5. List 5 critical assumptions I should validate
```

```
Format your response as structured bullet points with clear reasoning for each point.
```

2. Competitive Analysis Prompt

```
Act as a market research analyst. For my product idea: [PRODUCT_DESCRIPTION]

Research and provide:
1. 5 direct competitors with their key features
2. 3 indirect competitors or alternative solutions
3. Market gaps or underserved niches
4. Differentiation opportunities
5. Potential barriers to entry

Include specific examples and actionable insights for positioning.
```

3. User Persona Development Prompt

```
You are a UX researcher. Based on my product concept: [CONCEPT]

Create 3 distinct user personas including:
- Demographics and psychographics
- Core needs and pain points
- Current solutions they use
- Decision-making process
- Success criteria for adopting new tools

Focus on actionable insights that will guide feature development.
```

4. Feature Prioritization Prompt

```
You are a product manager with 10 years experience. For my app idea: [IDEA]

Help me prioritize features using the MoSCoW method:
- Must Have: Essential for MVP
- Should Have: Important but not critical
- Could Have: Nice additions
- Won't Have: Out of scope for v1

Consider development complexity, user value, and time-to-market. Provide reasoning for eac
```

YAML Configuration for Ideation Phase

```
# ideation-config.yaml
phase: "ideation"

research_framework:
  problem_validation:
    - customer_interviews: 10
    - survey_responses: 50
    - market_research_hours: 20

  success_metrics:
    - problem_market_fit_score: "&gt; 7/10"
    - user_interview_validation: "&gt; 80%"
    - competitor_gap_identified: true

ai_tools:
  research: ["perplexity", "consensus"]
  analysis: ["chatgpt-4", "claude-3"]
  validation: ["typeform", "calendly"]

deliverables:
  - problem_statement.md
  - user_personas.md
  - competitive_analysis.md
  - feature_roadmap.yaml
  - success_metrics.md

next_phase_criteria:
  - clear_problem_definition: true
  - target_user_identified: true
  - core_features_prioritized: true
  - technical_feasibility_assessed: true
```

Agents.md **Configuration for Ideation**

```
# AGENTS.md for Ideation Phase

## Role: Research and Validation Assistant

You are helping a solo developer validate and structure their product idea. Your goal is t

### Guidelines:
- Always ask clarifying questions before providing analysis
- Provide specific, actionable recommendations
- Include relevant examples and case studies
- Focus on validation over validation bias
- Challenge assumptions constructively

### Research Standards:
- Cite sources when providing market data
- Use multiple perspectives for competitive analysis
- Focus on user needs over feature lists
- Prioritize evidence-based insights

### Output Format:
- Use structured markdown with clear sections
- Include actionable next steps
- Provide confidence levels for recommendations
- Suggest specific validation methods
```

Phase 2: PROOF OF CONCEPT (POC) - Technical Validation

Objective

Validate core technical assumptions and demonstrate feasibility with minimal resource investment.

Key Activities

- Core algorithm/feature validation
- Technology stack selection
- Architecture planning
- Risk assessment
- Performance baseline establishment

AI Prompts for POC Phase

1. Technical Architecture Prompt

```
You are a senior software architect. I need to build a POC for: [DETAILED_FEATURE_DESCRIPT

Design a minimal technical architecture including:
1. Recommended technology stack with justification
2. Core components and their interactions
3. Data flow and storage requirements
4. Third-party services needed
5. Potential technical risks and mitigation strategies
6. Estimated development time for POC

Keep it simple but scalable. Focus on proving the core concept works.
```

2. POC Scope Definition Prompt

```
You are a technical project manager. For my product concept: [CONCEPT]

Help me define a focused POC that:
1. Tests the riskiest technical assumptions
2. Demonstrates core value proposition
3. Can be built in 1-2 weeks by a solo developer
4. Provides clear success/failure criteria
5. Sets up the foundation for MVP expansion

What should I build, and what should I deliberately exclude?
```

3. Code Structure Planning Prompt

```
You are an experienced developer. I'm building a POC for: [FEATURE_DESCRIPTION]

Generate:
1. Project folder structure
2. Key files and their purposes
3. Basic code templates for core functions
4. Testing approach for validation
5. Documentation structure

Use [TECHNOLOGY_STACK]. Focus on clean, readable code that can evolve into production.
```

4. POC Validation Prompt

```
You are a startup mentor. I've built a POC that: [POC_DESCRIPTION]

Help me evaluate:
1. What does this prove about technical feasibility?
2. What assumptions are still unvalidated?
3. What should I measure to determine success?
4. What are the key risks for scaling?
5. Should I proceed to MVP or iterate on POC?

Provide specific metrics and next-step recommendations.
```

YAML Configuration for POC Phase

```yaml
# poc-config.yaml
phase: "proof_of_concept"

technical_requirements:
  core_features:
    - feature_1: "authentication system"
    - feature_2: "data processing pipeline"
    - feature_3: "basic UI interface"

  technology_constraints:
    - budget: "$100/month max"
    - timeline: "2 weeks"
    - complexity: "minimal viable"

validation_criteria:
  technical_feasibility:
    - core_algorithm_works: true
    - performance_baseline: "&lt; 2s response"
    - data_integrity: "100%"

  business_validation:
    - user_testing_sessions: 5
    - feedback_score: "&gt; 3/5"
    - technical_debt_acceptable: true
```

```
tools_stack:
  frontend: "react"
  backend: "node.js"
  database: "sqlite"
  hosting: "vercel"
  ai_assistant: "cursor_ide"

success_metrics:
  - functionality_works: true
  - code_maintainable: true
  - scalability_path_clear: true
  - user_can_complete_core_task: true

next_phase_trigger:
  all_success_metrics: true
  stakeholder_approval: true
  resource_availability: true
```

Agents.md **Configuration for POC**

```
# AGENTS.md for POC Phase

## Role: Technical Implementation Assistant

You are a senior developer helping build a focused proof of concept. Prioritize working co

### Development Guidelines:
- Write clean, commented code
- Focus on core functionality only
- Use established patterns and libraries
- Include basic error handling
- Plan for easy iteration and extension

### Code Standards:
- Follow language-specific best practices
- Use meaningful variable and function names
- Include basic documentation
- Implement minimal but effective testing
- Consider security basics

### Problem-Solving Approach:
1. Understand the core requirement
2. Suggest the simplest working solution
3. Identify potential issues early
4. Provide alternatives when needed
5. Explain trade-offs clearly

### POC-Specific Focus:
- Prove technical feasibility
- Validate key assumptions
- Build foundation for MVP
- Document learnings and decisions
```

Phase 3: MVP - Minimum Viable Product

Objective

Build a functional product with core features that real users can test and provide feedback on.

Key Activities

- Feature development and integration
- User interface design and implementation
- Basic analytics and monitoring
- User testing and feedback collection
- Iterative improvement cycles

AI Prompts for MVP Phase

1. MVP Feature Implementation Prompt

```
You are a full-stack developer. I'm building an MVP with these validated features: [FEATUR

For each feature, provide:
1. Detailed implementation approach
2. Code structure and key components
3. User interface requirements
4. Data models and relationships
5. Testing strategy
6. Potential edge cases and handling

Focus on production-ready code that's maintainable and scalable.
```

2. User Experience Design Prompt

```
You are a UX designer. My MVP needs to serve these user journeys: [USER_JOURNEYS]

Design:
1. User flow diagrams for core tasks
2. Key screen layouts and components
3. Navigation structure
4. Error states and edge cases
5. Onboarding sequence
6. Success metrics for each flow

Keep it simple but effective for testing core value propositions.
```

3. MVP Testing Strategy Prompt

```
You are a QA engineer. For my MVP: [MVP_DESCRIPTION]

Create a comprehensive testing plan:
1. Unit tests for core business logic
2. Integration tests for key workflows
3. User acceptance criteria
4. Performance testing approach
5. Security testing basics
6. User testing script and scenarios

Focus on catching issues that would impact user adoption.
```

4. MVP Launch Preparation Prompt

```
You are a product launch specialist. My MVP is ready for initial user testing: [MVP_DETAIL

Help me prepare:
1. Launch checklist and timeline
2. User recruitment strategy
3. Feedback collection system
4. Success metrics and KPIs
5. Risk mitigation plan
6. Iteration planning process

Goal is to gather actionable user feedback for product-market fit validation.
```

YAML Configuration for MVP Phase

```yaml
# mvp-config.yaml
phase: "minimum_viable_product"

feature_set:
  core_features:
    - user_authentication: "required"
    - main_functionality: "required"
    - basic_dashboard: "required"

  nice_to_have:
    - advanced_analytics: "optional"
    - social_features: "optional"
    - mobile_optimization: "optional"

quality_standards:
  code_coverage: "&gt; 70%"
  performance:
    - load_time: "&lt; 3 seconds"
    - uptime: "&gt; 95%"

  user_experience:
    - usability_score: "&gt; 4/5"
    - task_completion_rate: "&gt; 80%"

launch_criteria:
  technical_readiness:
    - all_core_features_working: true
    - basic_security_implemented: true
    - error_handling_complete: true
    - monitoring_in_place: true

  business_readiness:
    - user_testing_completed: true
    - feedback_system_ready: true
    - support_process_defined: true

analytics_tracking:
  user_behavior:
    - feature_usage
    - user_journeys
    - drop_off_points

  business_metrics:
    - user_acquisition
    - retention_rate
    - feature_adoption

feedback_collection:
  methods: ["in_app_surveys", "user_interviews", "analytics"]
  frequency: "weekly"
  response_target: "20+ users"
```

Agents.md **Configuration for MVP**

```
# AGENTS.md for MVP Phase

## Role: Full-Stack Development Assistant

You are an experienced product developer helping build a production-ready MVP. Balance spe

### Development Standards:
- Write production-quality code
- Implement comprehensive error handling
- Include user-friendly error messages
- Follow security best practices
- Design for scalability and maintenance

### Feature Implementation:
- Focus on user value and experience
- Implement features completely, not partially
- Include proper validation and sanitization
- Consider performance implications
- Plan for analytics and monitoring

### Quality Assurance:
- Write meaningful tests for core functionality
- Validate user inputs thoroughly
- Handle edge cases gracefully
- Test across different scenarios
- Document known limitations

### User-Centric Approach:
- Prioritize intuitive user experiences
- Provide clear feedback and guidance
- Optimize for common user paths
- Handle errors gracefully with helpful messages
- Enable easy user feedback collection
```

Phase 4: DEPLOYMENT - Production Release

Objective

Deploy the MVP to production with proper monitoring, scaling, and maintenance processes.

Key Activities

- Production environment setup
- CI/CD pipeline implementation
- Monitoring and analytics integration
- Security hardening
- Scaling and maintenance planning

AI Prompts for Deployment Phase

1. Production Infrastructure Prompt

```
You are a DevOps engineer. I need to deploy my MVP: [MVP_DESCRIPTION] to production.

Design a deployment strategy including:
1. Hosting platform recommendation and setup
2. Domain and SSL certificate configuration
3. Database production configuration
4. Environment variables and secrets management
5. Backup and disaster recovery plan
6. Monitoring and alerting setup

Focus on cost-effective, reliable solutions for a solo developer.
```

2. CI/CD Pipeline Prompt

```
You are a DevOps specialist. Help me set up automated deployment for: [PROJECT_DETAILS]

Create:
1. GitHub Actions workflow for automated testing
2. Deployment pipeline for staging and production
3. Environment-specific configuration management
4. Rollback procedure and safety checks
5. Monitoring integration for deployment health
6. Documentation for the deployment process

Make it robust but simple enough for solo developer management.
```

3. Monitoring and Analytics Prompt

```
You are a Site Reliability Engineer. For my production MVP: [MVP_DESCRIPTION]

Set up comprehensive monitoring:
1. Application performance monitoring (APM)
2. Error tracking and alerting
3. User analytics and behavior tracking
4. Server health and resource monitoring
5. Security monitoring basics
6. Dashboard for key metrics

Focus on actionable insights that help improve the product and user experience.
```

4. Scaling and Maintenance Prompt

```
You are a technical consultant. My MVP is live with [USER_COUNT] users: [MVP_DETAILS]

Help me plan for growth:
1. Identify potential bottlenecks and scaling points
2. Performance optimization opportunities
3. Infrastructure scaling strategy
4. Maintenance and update procedures
5. Security hardening checklist
6. Cost optimization recommendations

Provide a roadmap for sustainable growth management.
```

YAML Configuration for Deployment Phase

```yaml
# deployment-config.yaml
phase: "production_deployment"

infrastructure:
  hosting:
    platform: "vercel" # or "railway", "render"
    region: "us-east-1"
    scaling: "automatic"

  database:
    provider: "planetscale" # or "supabase"
    backup_frequency: "daily"
    retention: "30_days"

  cdn: "cloudflare"
  monitoring: "sentry"

deployment_pipeline:
  triggers:
    - push_to_main: "production"
    - push_to_develop: "staging"

  tests:
    - unit_tests: "required"
    - integration_tests: "required"
    - performance_tests: "optional"

  safety_checks:
    - code_quality: "sonarcloud"
    - security_scan: "snyk"
    - dependency_audit: "npm_audit"

monitoring_stack:
  performance: "vercel_analytics"
  errors: "sentry"
  uptime: "uptime_robot"
  user_analytics: "posthog"

  alerts:
    - error_rate_threshold: "&gt; 5%"
    - response_time_threshold: "&gt; 5s"
    - downtime: "immediate"

security_measures:
  ssl: "automatic"
  headers:
    - csp: "strict"
    - hsts: "enabled"
    - x_frame_options: "deny"

  authentication:
    - rate_limiting: "enabled"
    - password_policy: "strong"
    - session_management: "secure"

maintenance:
  updates:
    - dependency_updates: "weekly"
    - security_patches: "immediate"
    - feature_releases: "bi_weekly"

  backups:
    - database: "daily"
    - code: "git_based"
    - assets: "weekly"
```

Agents.md Configuration for Deployment

```
# AGENTS.md for Deployment Phase

## Role: DevOps and Production Assistant

You are a senior DevOps engineer helping deploy and maintain production applications. Focu

### Infrastructure Guidelines:
- Choose managed services over self-hosted when possible
- Implement automated deployments and rollbacks
- Set up comprehensive monitoring from day one
- Plan for scaling before you need it
- Prioritize security without overcomplicating

### Deployment Standards:
- Zero-downtime deployments when possible
- Proper environment separation (dev/staging/prod)
- Automated testing in deployment pipeline
- Easy rollback mechanisms
- Clear deployment documentation

### Monitoring Philosophy:
- Monitor what matters for user experience
- Set up alerts that are actionable
- Track business metrics alongside technical metrics
- Plan for both success and failure scenarios
- Document incident response procedures

### Maintenance Approach:
- Automate routine maintenance tasks
- Keep dependencies up to date
- Regular security audits and updates
- Performance optimization based on real usage
- Plan for sustainable long-term operations
```

Integration and Governance Framework

Cascading Agents.md Structure

The repository implements a hierarchical configuration system where each phase inherits and extends the base governance rules:

```
repository/
├── agents.md                         # Base governance rules
├── phases/
│   ├── ideation/
│   │   ├── agents.md           # Inherits base + ideation-specific
│   │   └── config.yaml
│   ├── poc/
│   │   ├── agents.md           # Inherits base + poc-specific
│   │   └── config.yaml
│   ├── mvp/
│   │   ├── agents.md           # Inherits base + mvp-specific
│   │   └── config.yaml
│   └── deployment/
│       ├── agents.md           # Inherits base + deployment-specific
│       └── config.yaml
└── tools/
    ├── watcher-cli/            # Phase management utilities
    └── validation/             # Quality gates and checks
```

Watcher Role and Rituals

The "watcher" follows structured daily and phase-transition rituals:

Daily Watcher Ritual

1. **Morning Check (5 minutes)**
    - Review previous day's progress
    - Check AI-generated suggestions and feedback
    - Prioritize today's tasks based on phase objectives
    - Identify potential blockers
2. **Development Session**
    - Use phase-appropriate AI prompts for guidance
    - Follow agents.md guidelines for current phase
    - Document decisions and learnings
    - Update YAML configurations as needed
3. **Evening Review (10 minutes)**
    - Assess progress against phase criteria
    - Update project documentation
    - Plan next day's priorities
    - Commit code with meaningful messages

Phase Transition Ritual

1. **Completion Validation**
    - Check all YAML success criteria
    - Review deliverables against phase objectives
    - Gather stakeholder feedback if applicable

2. **Next Phase Preparation**
    - Update repository structure for new phase
    - Configure phase-specific agents.md and YAML
    - Set up tools and environments needed
    - Define success metrics for upcoming phase

Success Metrics and Validation

Each phase includes specific success criteria that must be met before progression:

- **Ideation → POC**: Clear problem definition, user validation, technical feasibility confirmed
- **POC → MVP**: Core technical assumptions validated, architecture proven, stakeholder buy-in
- **MVP → Deployment**: Feature complete, user tested, quality standards met, deployment ready
- **Deployment → Iteration**: Production stable, monitoring active, user feedback collected

This framework provides a structured, AI-assisted path from idea to production while maintaining the flexibility and learning focus essential for solo developers building with emerging AI tools.

This framework represents a practical, tested approach to AI-assisted solo development that scales from individual contributors to small teams while maintaining governance and quality standards necessary for sustainable software development.