ARC Development Planning & Roadmap

Strategic development plan for the Agentic Renovation Crew IDE

🎯 Project Vision

ARC (Agentic Renovation Crew) aims to be the first true IDE designed specifically for orchestrating multiple AI agents in a collaborative development environment. Our vision is to create a seamless interface where specialized AI agents work together to handle different aspects of software development, documentation, analysis, and project management.

🏗️ Architecture Decisions

Electron MVP First Approach

Decision: Build Electron version first, then fork to Tauri
Rationale:
- Faster initial development with familiar web technologies
- Rich ecosystem of Node.js packages for AI integration
- Mature Electron tooling and debugging capabilities
- Easier integration with local AI models (Ollama, LM Studio)

Technology Stack

- **Frontend**: React 18 + TypeScript
- **State Management**: Zustand for simplicity and performance
- **Styling**: CSS-in-JS with Framer Motion for animations
- **Desktop Framework**: Electron (MVP) → Tauri (future)
- **Build Tool**: Vite for fast development

📋 Development Phases

Phase 1: Foundation (Current - v1.0)

Timeline: 4-6 weeks
Status: 🔄 In Progress

Core Infrastructure

- [x] Electron application shell with React renderer
- [x] Type-safe IPC communication between main and renderer
- [x] Zustand stores for state management (App, Agent, Document)
- [x] Basic UI components and layout system
- [x] Mock agent system with streaming responses

Agent Management System

- [x] Agent configuration and lifecycle management
- [x] Multi-agent type support (code-gen, docs, discovery, etc.)
- [x] Agent enabling/disabling with performance metrics

- [x] Conversation management with message threading

UI/UX Implementation

- [x] Modern dark theme with gradient accents
- [x] Responsive sidebar with collapsible navigation
- [x] Chat interface with markdown rendering
- [x] Agent manager with configuration panels
- [x] Document manager with hierarchical organization
- [x] Settings panel with system information

Phase 2: AI Integration (v1.1-1.3)

Timeline: 6-8 weeks
Status: 📋 Planned

Local AI Model Support

- [ ] **Ollama Integration**: Connect to local Ollama server
- REST API client for model management
- Streaming chat completions
- Model switching and configuration
- [ ] **LM Studio Integration**: Connect to LM Studio local API
- OpenAI-compatible endpoint integration
- Model loading and configuration
- [ ] **node-llama-cpp Integration**: Direct local LLM integration
- GGML model loading
- CPU/GPU optimization
- Memory management

API Bridge Enhancement

- [ ] Provider-agnostic API abstraction layer
- [ ] Connection pooling and request queuing
- [ ] Error handling and retry mechanisms
- [ ] Performance monitoring and metrics

Agent Intelligence

- [ ] Context-aware prompt engineering
- [ ] Agent memory and conversation history
- [ ] Inter-agent communication protocols
- [ ] Learning and adaptation mechanisms

Phase 3: Advanced Agent Capabilities (v1.4-1.6)

Timeline: 8-10 weeks
Status: 📋 Planned

Multi-Agent Orchestration

- [ ] **Hierarchical Agents**: Chain of command structures
- [ ] **Goal-Based Agents**: Autonomous task completion
- [ ] **Reactive Agents**: Event-driven responses
- [ ] **Learning Agents**: Adaptive behavior improvement

Specialized Agent Types

- [ ] **Code Review Agent**: Automated code analysis and suggestions
- [ ] **Testing Agent**: Test generation and execution
- [ ] **Refactoring Agent**: Code improvement and optimization
- [ ] **Security Agent**: Vulnerability scanning and fixes
- [ ] **Performance Agent**: Optimization recommendations

Agent Collaboration

- [ ] Multi-agent workflows and pipelines
- [ ] Task delegation and result aggregation
- [ ] Conflict resolution between agent suggestions
- [ ] Collaborative decision making

Phase 4: Document Processing & Organization (v1.7-1.9)

Timeline: 6-8 weeks
Status: 📋 Planned

Advanced Document Management

- [ ] **Obsidian Vault Integration**: Seamless vault synchronization
- [ ] **Hierarchical Organization**: Smart categorization and tagging
- [ ] **Search and Discovery**: Semantic search across documents
- [ ] **Version Control**: Document history and change tracking

Content Processing

- [ ] **Text-to-Speech (TTS)**: Document audio generation
- [ ] **Speech-to-Text (STT)**: Voice note transcription
- [ ] **Image-to-Text (OCR)**: Document digitization
- [ ] **Video-to-Text**: Meeting and tutorial transcription

Format Support

- [ ] **Markdown**: Enhanced editing with live preview
- [ ] **PDF**: Text extraction and annotation
- [ ] **JSON Schemas**: Validation and documentation
- [ ] **Code Files**: Syntax highlighting and analysis

Phase 5: Tauri Migration (v2.0)

Timeline: 10-12 weeks
Status: 🔮 Future

Tauri Advantages

- **Performance**: Rust-based backend with smaller memory footprint
- **Security**: Enhanced security model with capability-based permissions
- **Distribution**: Smaller bundle sizes and better cross-platform support
- **Integration**: Better OS-level integration and native performance

Migration Strategy

- [ ] Gradual component migration starting with core utilities
- [ ] Parallel development to maintain Electron version compatibility
- [ ] Performance benchmarking and comparison

- [ ] User migration tools and documentation

🤖 Agent Architecture Deep Dive

Agent Type Taxonomy

1. Reactive Agents

```
interface ReactiveAgent {
  type: 'simple-reflex' | 'model-based-reflex'
  sensors: SensorInput[]
  actuators: ActionOutput[]
  conditionActionRules: Rule[]
}
```

- **Simple Reflex**: Direct stimulus-response patterns
- **Model-Based Reflex**: World model for informed decisions

2. Goal-Based Agents

```
interface GoalBasedAgent {
  goals: Goal[]
  actions: Action[]
  searchStrategy: SearchStrategy
  evaluationFunction: (state: State) => number
}
```

- **Problem-Solving**: Path finding to achieve goals
- **Planning**: Multi-step action sequences

3. Learning Agents

```
interface LearningAgent {
  learningElement: LearningAlgorithm
  performanceElement: PerformanceMetrics
  critic: CriticFunction
  problemGenerator: ProblemGenerator
}
```

- **Adaptive Behavior**: Improvement through experience
- **Performance Optimization**: Self-tuning parameters

4. Multi-Agent Systems

```
interface MultiAgentSystem {
  agents: Agent[]
  communicationProtocol: Protocol
  coordinationMechanism: CoordinationStrategy
  negotiationFramework: NegotiationRules
}
```

- **Cooperative**: Working toward common goals
- **Competitive**: Resource competition and optimization
- **Hierarchical**: Command and control structures

Agent Communication Protocols

Message Passing

```
interface AgentMessage {
  id: string
  sender: AgentId
  receiver: AgentId | 'broadcast'
  type: 'request' | 'response' | 'inform' | 'negotiate'
  content: any
  timestamp: number
  priority: number
}
```

Coordination Mechanisms

- **Blackboard Architecture**: Shared knowledge space
- **Contract Net Protocol**: Dynamic task allocation
- **Consensus Algorithms**: Distributed decision making

🎨 UI/UX Design Philosophy

Design Principles

1. **Agent-Centric**: UI designed around agent interactions
2. **Context Awareness**: Relevant information at the right time
3. **Progressive Disclosure**: Complex features revealed gradually
4. **Unified Experience**: Seamless flow between different functions

Visual Design

- **Dark Theme**: Reduces eye strain for long development sessions
- **Gradient Accents**: Modern visual appeal with purple/blue gradients
- **Smooth Animations**: Framer Motion for polished interactions
- **Information Density**: Optimal balance of data and whitespace

Interaction Patterns

- **Conversational Interface**: Chat-like interactions with agents
- **Direct Manipulation**: Drag-and-drop for document organization
- **Keyboard Shortcuts**: Power user efficiency features
- **Context Menus**: Right-click actions throughout the interface

🔧 Technical Architecture

Process Separation

```
┌─────────────────┐     IPC     ┌─────────────────┐
│   Main Process  │ ◄─────────► │ Renderer Process │
│                 │             │                 │
│ • File System   │             │ • React UI      │
│ • AI Integration│             │ • Zustand Store │
│ • IPC Handlers  │             │ • Agent Logic   │
│ • System APIs   │             │ • User Interaction│
└─────────────────┘             └─────────────────┘
```

State Management Strategy

- **Global State**: Application-wide settings and system information
- **Agent State**: Agent configurations, conversations, and metrics
- **Document State**: File system representation and metadata
- **UI State**: View modes, selections, and temporary state

Performance Considerations

- **Lazy Loading**: Components and data loaded on demand
- **Virtual Scrolling**: Efficient handling of large conversation histories
- **Memory Management**: Proper cleanup of AI model resources
- **Caching Strategy**: Intelligent caching of responses and documents

🚀 Deployment & Distribution

Build Pipeline

```
# Development
npm run electron:dev    # Hot-reload development

# Testing
npm run test           # Unit and integration tests
npm run test:e2e       # End-to-end testing

# Production
npm run build:app      # Full application build
npm run dist          # Platform-specific distributables
```

Platform Support

- **Windows**: NSIS installer with auto-updates
- **macOS**: DMG with notarization
- **Linux**: AppImage and Debian packages

Auto-Update Strategy

- Electron’s built-in updater for seamless updates
- Staged rollouts for stability
- Rollback capability for critical issues

📊 Success Metrics & KPIs

User Engagement

- Daily/Monthly Active Users (DAU/MAU)
- Session duration and frequency
- Feature adoption rates
- Agent usage patterns

Performance Metrics

- Application startup time
- Response latency for agent interactions
- Memory and CPU usage optimization
- Crash rates and stability metrics

AI Integration Success

- Agent response quality ratings
- Task completion rates
- User satisfaction with AI assistance
- Accuracy of agent recommendations

🤝 Community & Ecosystem

Open Source Strategy

- MIT license for maximum accessibility
- Clear contribution guidelines
- Regular community updates and roadmaps
- Plugin architecture for extensibility

Developer Experience

- Comprehensive documentation
- Video tutorials and examples
- Community forums and support channels
- Regular developer meetups and demos

📅 Next Steps

Immediate Priorities (Next 2 weeks)

1. ✅ Complete mock agent system implementation
2. ✅ Finalize UI component library and styling
3. 🔄 Comprehensive testing and bug fixes
4. 🔄 Documentation completion
5. 📋 Initial Ollama integration planning

Medium Term (Next 4-6 weeks)

1. Begin Ollama integration development

2. Implement agent performance metrics
3. Add document import/export functionality
4. Create comprehensive test suite
5. User feedback collection and iteration

Long Term (Next 6 months)

1. Full local AI model support across providers
2. Multi-agent collaboration features
3. Advanced document processing capabilities
4. Performance optimization and scalability
5. Tauri migration preparation

This planning document is a living document that evolves with our understanding and user feedback. Regular updates ensure we stay aligned with our vision while remaining responsive to user needs.

ARC Development Team 🚀
Building the Future of AI-Orchestrated Development