agent-architecture

Custom Agent Architecture

This diagram illustrates the architecture and components of custom GitHub Copilot agents.

Agent Definition Structure

```
graph TB
    subgraph "Agent Definition File (.agent.md)"
        FM[Front Matter<br/>YAML Metadata]
        Body[Markdown Body<br/>Instructions]
    end

    FM --> Name[name: agent-name]
    FM --> Desc[description: purpose]
    FM --> Tools[tools: capabilities]
    FM --> Model[model: AI model]

    Body --> Identity[Identity & Role]
    Body --> Resp[Responsibilities]
    Body --> Context[Context]
    Body --> Constraints[Constraints]
    Body --> Process[Analysis Process]
    Body --> Output[Output Format]
    Body --> Tone[Tone & Style]

    style FM fill:#e1e8ff
    style Body fill:#f0f0f0
```

Agent Component Hierarchy

```
graph LR
    Agent[Custom Agent] --> Core[Core Definition]
    Agent --> Behavior[Behavioral Rules]
    Agent --> Interface[Interface Spec]

    Core --> ID[Identity/Role]
    Core --> RESP[Responsibilities]
    Core --> CTX[Context/Domain]

    Behavior --> CONST[Constraints<br/>ALWAYS/NEVER]
    Behavior --> PROC[Process Steps]
    Behavior --> TONE[Tone Guidelines]

    Interface --> IN[Input Expectations]
    Interface --> OUT[Output Format]
    Interface --> EX[Examples]

    style Agent fill:#4a90e2,color:#fff
    style Core fill:#e1f5e1
    style Behavior fill:#fff4e1
    style Interface fill:#ffe1e1
```

Agent Execution Flow

```
sequenceDiagram
    participant U as User
    participant UI as VS Code UI
    participant A as Selected Agent
    participant LLM as Language Model
    participant R as Repository

    U->>UI: Select agent from dropdown
    U->>A: Provide prompt/request
    A->>A: Load agent definition
    A->>A: Merge agent instructions<br/>with user prompt
    A->>R: Access repository context
    A->>LLM: Send enriched prompt

    loop Reasoning
        LLM->>LLM: Apply agent constraints
        LLM->>LLM: Follow defined process
        LLM->>LLM: Structure output per format
    end

    LLM->>A: Generate response
    A->>A: Validate against format
    A->>UI: Display structured result
    UI->>U: Present agent output
```

Agent Instruction Processing

```
graph TB
    Start[User Request] --> Load[Load Agent Definition]
```

```
    Load --> Merge[Merge Instructions]

    subgraph "Instruction Layers"
        Base[Base Copilot Behavior]
        Global[Copilot Instructions<br/>.github/copilot-instructions.md]
        Agent[Agent-Specific Rules<br/>.github/agents/name.agent.md]
        Prompt[User Prompt]
    end

    Merge --> Stack[Instruction Stack]
    Stack --> Base
    Base --> Global
    Global --> Agent
    Agent --> Prompt

    Prompt --> Execute[Execute with LLM]
    Execute --> Response[Structured Response]

    style Agent fill:#4a90e2,color:#fff
    style Stack fill:#f0f0f0
```

Agent Storage and Discovery

```
graph TB
    Repo[Repository] --> GitHub[.github/]
    GitHub --> Agents[agents/]

    Agents --> README[README.md<br/>Catalog/Index]
    Agents --> Agent1[architecture-reviewer.agent.md]
    Agents --> Agent2[backlog-generator.agent.md]
    Agents --> Agent3[test-strategist.agent.md]
    Agents --> AgentN[custom-agent.agent.md]

    VSCode[VS Code Extension] -.Scans.-> Agents
    VSCode --> Dropdown[Agent Dropdown Menu]

    User[Developer] --> Dropdown
    Dropdown --> Selection[Select Agent]
    Selection --> Agent1
    Selection --> Agent2
    Selection --> Agent3
    Selection --> AgentN

    style Agents fill:#e1e8ff
    style VSCode fill:#4a90e2,color:#fff
```

Agent Lifecycle

```
stateDiagram-v2
    [*] --> Design: Identify Need
    Design --> Draft: Create Definition
    Draft --> Test: Write Instructions
    Test --> Refine: Validate Behavior
    Refine --> Test: Issues Found
    Refine --> Review: Success Criteria Met
    Review --> Production: PR Approved
```

```
    Production --> Monitor: In Use
    Monitor --> Update: Feedback Collected
    Update --> Test: Improve Instructions
    Production --> Retire: No Longer Needed
    Retire --> [*]

    note right of Design
        Define role, scope,
        success criteria
    end note

    note right of Test
        Test with real scenarios,
        iterate on instructions
    end note

    note right of Production
        Team usage,
        documented in catalog
    end note
```

Agent Types and Specializations

```
mindmap
    root((Custom Agents))
        Reviewers
            Architecture Reviewer
            Security Reviewer
            Code Quality Reviewer
            Performance Auditor
        Generators
            Backlog Generator
            Documentation Writer
            Test Generator
            Schema Designer
        Strategists
            Test Strategist
            Migration Planner
            Refactoring Advisor
        Analysts
            Dependency Analyzer
            Impact Analyzer
            Complexity Scorer
```

Multi-Agent Workflows

```
graph LR
    subgraph "Sequential Agent Chain"
        A1[Backlog Generator] --> Output1[User Stories]
        Output1 --> A2[Architecture Reviewer]
        A2 --> Output2[Design Feedback]
        Output2 --> A3[Test Strategist]
        A3 --> Output3[Test Plan]
    end

    subgraph "Parallel Agent Usage"
```

```
        Task[Feature Request]
        Task --> P1[Architecture Review]
        Task --> P2[Security Review]
        Task --> P3[Performance Review]
        P1 --> Merge[Consolidated Feedback]
        P2 --> Merge
        P3 --> Merge
    end

    style A1 fill:#e1f5e1
    style A2 fill:#fff4e1
    style A3 fill:#e1e8ff
```

Agent Composition Pattern

```
graph TB
    subgraph "Agent Definition"
        Meta[Metadata<br/>name, description, tools, model]
        Role[Role Definition<br/>Who is this agent?]
        Scope[Scope & Boundaries<br/>What does it do/not do?]
        Rules[Behavioral Rules<br/>Constraints, process, tone]
        Format[Output Specification<br/>Structure, sections, examples]
    end

    Meta --> Runtime[Runtime Behavior]
    Role --> Runtime
    Scope --> Runtime
    Rules --> Runtime
    Format --> Runtime

    Runtime --> Consistent[Consistent Results]
    Runtime --> Predictable[Predictable Format]
    Runtime --> Reliable[Reliable Guidance]

    style Runtime fill:#4a90e2,color:#fff
```

Key Architectural Principles

1. Separation of Concerns

- Agent definitions are separate from Copilot Instructions
- Each agent has a focused, well-defined role
- No overlap or duplication between agents

2. Composability

- Agents can be used in sequence
- Outputs from one agent can inform another
- Standard output formats enable chaining

3. Discoverability

- Agents stored in `.github/agents/`
- Consistent naming: `name.agent.md`

- README provides catalog and usage guide

4. Versioning

- Agent definitions are version controlled
- Changes tracked through git history
- Breaking changes documented

5. Extensibility

- Teams can add custom agents
- Agent template provides structure
- No limit to number of agents

See Also

- Lab 08: Agent Design
- Agent Design Guide
- Agent Governance
- Custom Agent Catalog