

lab-08-agent-design

- [Lab 08: Designing Effective Custom Agents](#)
 - [Objectives](#)
 - [Prerequisites](#)
 - [Background](#)
 - [Agents Are Products, Not Prompts](#)
 - [Core Components of Agent Instructions](#)
 - [1. Identity & Role](#)
 - [2. Responsibilities](#)
 - [3. Context](#)
 - [4. Constraints](#)
 - [5. Process/Approach](#)
 - [6. Output Format](#)
 - [7. Tone & Approach](#)
 - [Exercise 1: Analyze Existing Agents \(10 minutes\)](#)
 - [Instructions](#)
 - [Questions](#)
 - [Exercise 2: Iterating on Agent Instructions \(15 minutes\)](#)
 - [Scenario](#)
 - [Part A: Baseline Behavior](#)
 - [Part B: Refine the Agent](#)
 - [Part C: Re-test Behavior](#)
 - [Reflection Questions](#)
 - [Design Patterns for Reliable Agents](#)
 - [Pattern 1: Role-Based Scope](#)
 - [Pattern 2: Explicit Constraints](#)
 - [Pattern 3: Structured Outputs](#)
 - [Pattern 4: Educational Tone](#)
 - [Pattern 5: Boundaries & Disclaimers](#)
 - [Governance Considerations](#)
 - [Versioning](#)
 - [Review Process](#)
 - [Team Alignment](#)
 - [Documentation](#)
 - [Common Pitfalls](#)
 - ~~[Pitfall 1: Task-Based Agents](#)~~
 - ~~[Pitfall 2: Vague Instructions](#)~~
 - ~~[Pitfall 3: Over-Scoping](#)~~
 - ~~[Pitfall 4: Under-Testing](#)~~
 - ~~[Pitfall 5: No Iteration Loop](#)~~
 - [Key Takeaways](#)
 - [Next Steps](#)
 - [Additional Resources](#)

Lab 08: Designing Effective Custom Agents

Module: 4

Duration: 25 minutes

Part: Advanced GitHub Copilot (Part 2)

Objectives

By the end of this lab, you will:

- Understand the key components of agent instructions
- Learn how to design agents around roles, not tasks
- Practice iterating on agent behavior through instruction refinement
- Recognize patterns for creating reliable, trustworthy agents

Prerequisites

- Completion of [Lab 07: Workflow Agents](#)
- VS Code with GitHub Copilot extension
- Access to the TaskManager workshop repository

Background

Agents Are Products, Not Prompts

Creating a custom agent isn't just writing a prompt. It's designing a **reusable product** that your team will rely on.

Key Principle: Design for roles (specialists), not tasks (one-off actions)

 **Task-based (Bad):** "Generate unit tests for a method"

 **Role-based (Good):** "Test Strategist - Proposes comprehensive test strategies"

Core Components of Agent Instructions

Every effective agent definition has:

1. Identity & Role

- Who is this agent?
- What expertise does it embody?

You are an expert software architect specializing in Clean Architecture...

2. Responsibilities

- What does this agent do?
- What is in scope vs out of scope?

`## Responsibilities`

- Analyze code structure for architectural boundary violations
- Identify dependency direction issues
- Review domain model design for DDD patterns

3. Context

- What does the agent need to know about the project?
- What standards, patterns, or constraints apply?

Context

This project follows Clean Architecture with these layers:

- Domain: Business logic (no external dependencies)
 - Application: Use cases (depends on Domain only)
- ...

4. Constraints

- What should the agent ALWAYS do?
- What should it NEVER do?

Constraints

- ALWAYS check for circular dependencies
- NEVER recommend breaking Clean Architecture boundaries

5. Process/Approach

- How should the agent work through the task?
- What steps should it follow?

Analysis Process

1. Identify which layer(s) the code belongs to
2. Check dependencies against allowed directions
3. Review domain modeling

6. Output Format

- How should results be structured?
- What sections or headings should appear?

Output Format

Provide your review in this structured format:

Architecture Review Summary

- **Scope:** [what was reviewed]
 - **Overall Assessment:** [Pass/Needs Attention/Refactor Required]
- ...

7. Tone & Approach

- How should the agent communicate?
- What's the personality or style?

Tone

- Be direct and constructive

- Explain WHY something is a concern (educational)
 - Acknowledge good practices when present
-

Exercise 1: Analyze Existing Agents (10 minutes)

Instructions

Open each of the three custom agents and map their components:

Architecture Reviewer

- **File:** .github/agents/architecture-reviewer.agent.md
- **Role:** [Identify the role]
- **Key Responsibilities:** [List 3]
- **Critical Constraints:** [List 2-3]
- **Output Structure:** [Describe the format]

Backlog Generator

- **File:** .github/agents/backlog-generator.agent.md
- **Role:** [Identify the role]
- **Key Responsibilities:** [List 3]
- **Critical Constraints:** [List 2-3]
- **Output Structure:** [Describe the format]

Test Strategist

- **File:** .github/agents/test-strategist.agent.md
- **Role:** [Identify the role]
- **Key Responsibilities:** [List 3]
- **Critical Constraints:** [List 2-3]
- **Output Structure:** [Describe the format]

Questions

- What patterns do you notice across all three agents?
 - Which component seems most critical for consistency?
 - Are there any missing components you'd add?
-

Exercise 2: Iterating on Agent Instructions (15 minutes)

Scenario

The **Test Strategist** agent sometimes provides too many tests, including low-value scenarios. You want to refine it to focus on **high-value tests only**.

Part A: Baseline Behavior

1. Open Copilot Chat in Agent Mode
2. Select **Test Strategist**
3. Prompt: Propose test scenarios for a simple getter method that returns a task's title

Observe: Does it over-test? Does it recommend unnecessary tests?

Part B: Refine the Agent

1. Open `.github/agents/test-strategist.agent.md`
2. Add this constraint to the **Constraints** section:
 - Focus on HIGH-VALUE tests only (avoid trivial getters/setters)
 - Skip tests for auto-implemented properties or simple pass-through methods
 - Prioritize tests that verify business logic, invariants, and edge cases

1. Save the file

Part C: Re-test Behavior

1. Return to Copilot Chat (Agent Mode)
2. Select **Test Strategist** again
3. Use the same prompt: Propose test scenarios for a simple getter method that returns a task's title

Observe: Did the agent's behavior change? Did it decline or simplify the recommendation?

Reflection Questions

1. Did the constraint reduce over-testing?
 2. What other refinements would improve this agent?
 3. How many iterations would you expect before an agent is "production-ready"?
-

Design Patterns for Reliable Agents

Pattern 1: Role-Based Scope

-  **Do:** "You are a code reviewer specializing in security"
 **Don't:** "Generate code for feature X"

Pattern 2: Explicit Constraints

-  **Do:** "NEVER recommend breaking layer boundaries"
 **Don't:** Leave implicit assumptions unstated

Pattern 3: Structured Outputs

- ✓ **Do:** Define sections, headings, and formats
- ✗ **Don't:** Allow free-form, unpredictable responses

Pattern 4: Educational Tone

- ✓ **Do:** "Explain WHY this is a concern"
- ✗ **Don't:** Just list issues without context

Pattern 5: Boundaries & Disclaimers

- ✓ **Do:** "This agent reviews; humans decide"
 - ✗ **Don't:** Imply the agent is authoritative
-

Governance Considerations

Versioning

- Agents should be versioned (like code)
- Track changes in git commit history
- Consider semantic versioning for major changes

Review Process

- Agent changes require **pull request review**
- Test agent behavior before merging
- Document breaking changes

Team Alignment

- Agents encode **team decisions**, not individual preferences
- Discuss agent behavior in retrospectives
- Update agents as practices evolve

Documentation

- Maintain a **catalog of agents** ([docs/guides/custom-agent-catalog.md](#))
 - Document when to use each agent
 - Provide examples of good vs bad usage
-

Common Pitfalls

✗ Pitfall 1: Task-Based Agents

Creating agents for single, one-off tasks instead of repeatable roles.

Fix: Design for workflows, not individual actions.

Pitfall 2: Vague Instructions

Leaving agent behavior open to interpretation.

Fix: Use explicit constraints and structured outputs.

Pitfall 3: Over-Scoping

Making agents do too much.

Fix: Keep agents focused on one role or domain.

Pitfall 4: Under-Testing

Deploying agents without validating behavior.

Fix: Test agents with real scenarios before sharing.

Pitfall 5: No Iteration Loop

Treating agents as "set and forget."

Fix: Continuously refine based on usage and feedback.

Key Takeaways

- Agents are products** - Design, test, and maintain them like code
 - Role-based design** - Specialists, not task executors
 - Explicit constraints** - State what the agent must/must not do
 - Structured outputs** - Consistency requires format
 - Iterate continuously** - Refine based on real usage
 - Govern as team assets** - Version, review, and document
-

Next Steps

In [Lab 09: Capstone - Build Your Own Agent](#), you'll **create a production-ready custom agent** from scratch, applying everything you've learned.

Additional Resources

- [Agent Design Guide](#)
- [Agent Governance](#)
- [Custom Agent Catalog](#)