lab-07-workflow-agents

Lab 07: Workflow Agents in Action

Module: 3
Duration: 45 minutes
Part: Advanced GitHub Copilot (Part 2)

Objectives

By the end of this lab, you will:

- Apply custom agents to real development workflows
- Compare agent outputs to standard Copilot Chat responses
- Evaluate reliability and consistency differences
- Understand when agents provide meaningful value over ad-hoc prompting

Prerequisites

- Completion of Lab 06: Custom Agents Intro

- VS Code with GitHub Copilot extension
- Access to the TaskManager workshop repository
- Familiarity with the three custom agents (Architecture Reviewer, Backlog Generator, Test Strategist)

Lab Structure

You'll work through **3 workflow scenarios**, each testing a different agent. For each scenario, you'll:

1. **First:** Use standard Copilot Chat (no agent)
2. **Second:** Use the appropriate custom agent
3. **Compare:** Document differences in quality, structure, and consistency

---

Scenario 1: Backlog Generation (15 minutes)

Context

Your team wants to add a **notification system** to the TaskManager application. Users should receive notifications when:

- A task is assigned to them
- A task deadline is approaching
- A task status changes

You need to break this down into user stories with acceptance criteria.

Part A: Standard Copilot Chat

Instructions:

1. Open Copilot Chat (no agent selected)
2. Use this prompt:

```
Create user stories for a notification system in the TaskManager app.
Users should get notifications when tasks are assigned, deadlines approach, or
status changes.
Include acceptance criteria.
```

1. **Record the output** (copy/paste into a document or note the structure)

Questions to consider:

- How are the stories formatted?
- Are acceptance criteria specific and testable?
- Is the output consistent with agile best practices?

Part B: Backlog Generator Agent

Instructions:

1. Switch to **Agent Mode**
2. Select **Backlog Generator** from the agent dropdown
3. Use the same (or similar) prompt:

```
Create user stories for a notification system in the TaskManager app.
Users should get notifications when tasks are assigned, deadlines approach, or
status changes.
```

1. **Record the output**

Expected agent behavior:

- User stories in "As a... I want... So that..." format
- Specific, testable acceptance criteria
- Story points or sizing estimates
- Dependencies identified
- INVEST principles applied

Comparison Questions

Aspect	Standard Chat	Backlog Generator Agent
Story Format	[Your observation]	[Your observation]
Acceptance Criteria Quality	[Your observation]	[Your observation]
Completeness	[Your observation]	[Your observation]
Consistency	[Your observation]	[Your observation]
Ready for Sprint Planning?	[Your observation]	[Your observation]

Reflection:

- Which output would you prefer to present to your product owner?
- Would the agent output save you revision time?
- How much manual cleanup is needed in each case?

Scenario 2: Architecture Review (15 minutes)

Context

A team member has submitted a pull request that adds a new `NotificationService` in the **Application** layer. You want to ensure it follows Clean Architecture and DDD patterns before approving.

Setup

Create a sample file to review (or use an existing Application service):

File: `src/TaskManager.Application/Services/NotificationService.cs`

```csharp
namespace TaskManager.Application.Services;

public class NotificationService
{
    private readonly ITaskRepository _taskRepository;
    private readonly IEmailService _emailService;

    public NotificationService(ITaskRepository taskRepository, IEmailService
          emailService)
    {
        _taskRepository = taskRepository;
        _emailService = emailService;
    }

    public async Task NotifyTaskAssignedAsync(int taskId, string
          assigneeEmail)
    {
        var task = await _taskRepository.GetByIdAsync(taskId);
        if (task != null)
        {
            await _emailService.SendAsync(assigneeEmail, "Task Assigned",
          $"You have been assigned task: {task.Title}");
        }
    }
}
```

Part A: Standard Copilot Chat

Instructions:

1. Open Copilot Chat (no agent)
2. Open the `NotificationService.cs` file
3. Prompt:

```
Review this NotificationService for Clean Architecture compliance and suggest
improvements.
```

1. **Record the feedback**

Part B: Architecture Reviewer Agent

Instructions:

1. Switch to **Agent Mode**
2. Select **Architecture Reviewer** from dropdown
3. Same prompt:

```
Review this NotificationService for Clean Architecture compliance and suggest
improvements.
```

1. **Record the feedback**

Expected agent behavior:

- Structured review (Strengths, Concerns, Violations, Recommendations)
- Layer-specific analysis (Application layer rules)
- DDD pattern evaluation
- Dependency direction checks
- Specific, actionable recommendations

Comparison Questions

Aspect	Standard Chat	Architecture Reviewer Agent
Review Structure	[Your observation]	[Your observation]
Depth of Analysis	[Your observation]	[Your observation]
Actionable Recommendations	[Your observation]	[Your observation]
Consistency with Standards	[Your observation]	[Your observation]
Ready to Use in PR Review?	[Your observation]	[Your observation]

Reflection:

- Did the agent identify issues standard chat missed?
- Is the agent's format more useful for code review comments?
- Would you trust this agent's review as a first pass?

Scenario 3: Test Strategy (15 minutes)

Context

You're implementing a new feature: **Task Assignment**. A task can be assigned to a user, and the assignment should:

- Validate that the user exists
- Validate that the task exists
- Record assignment timestamp
- Emit a domain event (`TaskAssigned`)

You need a test strategy.

Part A: Standard Copilot Chat

Instructions:

1. Open Copilot Chat (no agent)
2. Prompt:

```
Propose test scenarios for a task assignment feature.
Validate user and task exist, record timestamp, emit domain event.
```

1. **Record the test scenarios**

Part B: Test Strategist Agent

Instructions:

1. Switch to **Agent Mode**
2. Select **Test Strategist** from dropdown
3. Same prompt:

```
Propose test scenarios for a task assignment feature.
Validate user and task exist, record timestamp, emit domain event.
```

1. **Record the test scenarios**

Expected agent behavior:

- Categorized tests (unit, integration, edge cases)
- AAA pattern (Arrange, Act, Assert) descriptions
- Specific test names
- Edge cases and boundary conditions
- Error handling scenarios
- Testability recommendations

Comparison Questions

Aspect	Standard Chat	Test Strategist Agent
Test Organization	[Your observation]	[Your observation]
Coverage Completeness	[Your observation]	[Your observation]
Edge Case Identification	[Your observation]	[Your observation]
Test Naming Clarity	[Your observation]	[Your observation]
Ready to Implement?	[Your observation]	[Your observation]

Reflection:

- Did the agent identify edge cases you hadn't considered?
- Is the agent's categorization helpful?
- Would you feel confident implementing tests from the agent's output?

Group Discussion (If in Workshop Setting)

Share your findings:

1. **Which agent provided the most value?**
2. **Did agents catch issues that standard chat missed?**
3. **Were the structured outputs more useful than free-form responses?**
4. **What are the limitations of agents?**

Key Takeaways

Agents Excel At:

✅ **Structured, repeatable workflows** (reviews, planning, analysis)
✅ **Consistency across team members** (same agent = same format)
✅ **Encoding domain expertise** (architecture, testing, product practices)
✅ **First-pass automation** (reduce manual work)

Agents Are Not:

❌ **Always correct** - You're still accountable
❌ **Replacements for human judgment** - They're assistants
❌ **One-size-fits-all** - Use the right agent for the job

When Agents Shine:

- **Code reviews** (automated first pass)
- **Backlog grooming** (consistent story format)
- **Test planning** (comprehensive coverage)
- **Documentation generation** (structured outputs)
- **Knowledge transfer** (encode team practices)

Challenge Exercise (Optional)

Try this:

Create a **custom scenario** for your own work:

1. Identify a repetitive workflow in your daily work
2. Use standard Chat to perform it
3. Then use the most relevant agent (or imagine what agent you'd need)
4. Compare the results

Bonus: Draft an agent definition for a workflow you need. (You'll formalize this in Labs 08-09.)

Next Steps

In Lab 08: Agent Design, you'll learn **how to design effective agents** — instruction components, iteration loops, and governance.

Additional Resources

- Agent Scenario Examples
- Custom Agent Catalog
- Agent Workflow Patterns Diagram