advanced-github-copilot

Advanced GitHub Copilot

Custom Agents & AI-Driven Development Workflows

Duration: 3 Hours
Format: Instructor-led, hands-on

Welcome to Part 2

What You'll Learn

- Understand **Ask, Edit, and Agent** interaction models
- Use **Custom Copilot Agents** for repeatable workflows
- **Design effective agents** for your team
- Apply **governance** to AI usage
- **Build a production-ready agent** from scratch

Prerequisites

✅ Completion of Part 1 (or equivalent Copilot experience)
✅ VS Code with GitHub Copilot extension
✅ Access to workshop repository
✅ Familiarity with code generation and refactoring workflows

Part 1 Recap (5 minutes)

In Part 1, you learned:

- Using Copilot for TDD and refactoring
- Copilot Instructions for guardrails
- Documentation automation
- Requirements-to-code workflows

In Part 2, we'll focus on:

Shaping **how teams work** using AI

Today's Journey

```
Module 0: Kickoff & Context Reset (10 min)
Module 1: Interaction Models (25 min)
Module 2: Custom Agents Intro (30 min)
Module 3: Workflow Agents (45 min)
Module 4: Agent Design (25 min)
Module 5: Capstone Lab (35 min)
Module 6: Wrap-Up & Governance (10 min)
```

Module 1

Copilot Interaction Models

Ask, Edit, Agent

Duration: 25 minutes

Three Ways to Interact with Copilot

Ask Mode

Purpose: Learning, exploration, explanation
Result: Answers, guidance (no changes)

Edit Mode

Purpose: Localized, scoped code changes
Result: Direct file modifications

Agent Mode

Purpose: Multi-step, repository-level workflows
Result: Planned changes with human checkpoints

Ask Mode: When to Use

✅ **Use when:**

- Understanding code or concepts
- Exploring options
- Learning patterns
- Getting explanations

❌ **Don't use when:**

- You need actual code changes
- Implementing features
- Refactoring across files

Edit Mode: When to Use

✅ **Use when:**

- Targeted, scoped changes
- You know exactly what to modify
- Single file or small set of files
- Quick fixes or localized refactors

❌ **Don't use when:**

- Changes span many files
- You need analysis first
- Exploring alternatives

Agent Mode: When to Use

✅ **Use when:**

- Multi-file workflows
- Repository-level analysis
- Complex refactoring
- Need plan-execute-review cycle

❌ **Don't use when:**

- Simple, quick edits
- Single file changes
- Learning or exploring

Key: Agent Mode = Human-in-the-loop by design

Demo: Same Task, Three Ways

Task: Add Priority property to Task entity

1. **Ask** → Explanation only
2. **Edit** → Local file change
3. **Agent** → Repository-wide analysis and plan

Observe the differences in:

- Scope
- Control
- Visibility
- Workflow

Key Takeaway

> Agent Mode is not "better chat"
> It's a **different execution model**

Use the **right mode** for the **right job**

Module 2

Custom Copilot Agents

Chat Participants as Specialists

Duration: 30 minutes

What Are Custom Agents?

Named chat participants with specific roles

- Selectable from agent dropdown
- Role-based AI personas (e.g., Architecture Reviewer)
- Defined scope and constraints
- Structured, consistent outputs
- Encode team knowledge

Mental Model: The Specialist

```
Standard Copilot Chat = General AI Assistant
```

```
Custom Agent = Domain Expert Consultant
```

You wouldn't ask a general assistant to:

- Review architecture → Ask an architect
- Plan testing → Ask a QA specialist
- Generate backlog → Ask a product analyst

Custom agents ARE those specialists

Agents vs Instructions vs Prompts

Feature	Instructions	Prompts	Agents
Scope	Always-on	One-off	On-demand
Purpose	Guardrails	Specific task	Workflow
Reusability	Implicit	Manual	Built-in
Consistency	Background	Variable	Structured

When to Use Custom Agents

✅ **Use agents for:**

- Repeated workflows (reviews, planning)
- Consistent outputs across team
- Encoding expert knowledge
- Validation and review tasks

❌ **Don't use agents for:**

- Simple one-off prompts
- Exploration or learning

- Unique, non-repeatable tasks

Workshop Agents

Architecture Reviewer

Reviews code for Clean Architecture & DDD compliance

Backlog Generator

Creates user stories with acceptance criteria

Test Strategist

Proposes comprehensive test strategies

Guided Exercise

Try the Architecture Reviewer agent:

1. Open Copilot Chat in Agent Mode
2. Select "Architecture Reviewer" from dropdown
3. Prompt: "Review the Task domain model"
4. Compare to standard Copilot Chat output

Observe:

- Structured format
- Consistency
- Depth of analysis

Key Insight

> If Copilot Instructions are guardrails,
> Custom agents are specialists you consult

Module 3

Workflow Agents in Action

Hands-On Lab

Duration: 45 minutes

Lab Overview

You'll apply agents to **3 real workflows:**

1. **Backlog Generation** (user stories)
2. **Architecture Review** (code quality)
3. **Test Strategy** (test planning)

For each:

- Use standard Chat first
- Then use the custom agent
- Compare quality and consistency

Workflow 1: Backlog Generation

Scenario: Add notification system to TaskManager

Compare:

- Standard Chat → Free-form stories
- Backlog Generator → Structured INVEST stories

Key question: Which output is sprint-ready?

Workflow 2: Architecture Review

Scenario: Review NotificationService for Clean Architecture

Compare:

- Standard Chat → General feedback
- Architecture Reviewer → Structured analysis

Key question: Which review is PR-ready?

Workflow 3: Test Strategy

Scenario: Propose tests for task assignment feature

Compare:

- Standard Chat → Basic scenarios
- Test Strategist → Comprehensive coverage

Key question: Which strategy is implementation-ready?

Discussion Questions

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

Agents Excel At

✅ Structured, repeatable workflows
✅ Consistency across team members
✅ Encoding domain expertise
✅ First-pass automation

But they're not:

❌ Always correct (you're accountable)
❌ Replacements for human judgment
❌ One-size-fits-all solutions

Module 4

Designing Effective Agents

Principles and Patterns

Duration: 25 minutes

Core Principle

> Agents are products, not prompts

Design, test, and maintain them **like code**

Agent Components

1. **Identity & Role** - Who is this agent?
2. **Responsibilities** - What does it do?
3. **Context** - What does it need to know?
4. **Constraints** - Rules it must follow
5. **Process** - How it approaches tasks
6. **Output Format** - Structured results
7. **Tone** - Communication style

Design Pattern: Role-Based Scope

✅ **Do:** "You are a code reviewer specializing in security"

❌ **Don't:** "Generate code for feature X"

Focus on WHO, not WHAT

Design Pattern: Explicit Constraints

✅ **Do:**

```
## Constraints
- ALWAYS check for circular dependencies
- NEVER recommend breaking layer boundaries
```

❌ **Don't:** Leave assumptions unstated

Design Pattern: Structured Outputs

✅ **Do:** Define sections and format

```
## Output Format

### Review Summary
- **Scope:** [what was reviewed]
- **Assessment:** [Pass/Needs Attention/Refactor]

### Findings
...
```

❌ **Don't:** Allow free-form responses

Iteration Loop

```
Define → Test → Observe → Refine → Repeat
```

Example refinement:

- Agent over-tests simple getters
- Add constraint: "Focus on high-value tests only"
- Re-test with same scenario
- Observe improved behavior

Governance Considerations

Versioning

- Track changes in git
- Semantic versioning for major updates

Review Process

- Agent changes require PR review
- Test before merging

Team Alignment

- Agents encode **team decisions**
- Update as practices evolve

Common Pitfalls

❌ **Task-based agents** → Use role-based
❌ **Vague instructions** → Be explicit
❌ **Over-scoping** → Keep focused
❌ **No testing** → Validate before sharing
❌ **Set-and-forget** → Iterate continuously

Module 5

Capstone Lab

Build Your Own Agent

Duration: 35 minutes

Capstone Overview

You will:

1. Select an agent role
2. Define success criteria
3. Create the agent definition
4. Test with real scenarios
5. Iterate and refine
6. Document for team use

Agent Role Options

- **Code Reviewer** - Quality and standards
- **Documentation Writer** - API docs, READMEs
- **Performance Auditor** - Optimization opportunities
- **Security Reviewer** - Vulnerability detection
- **Your Custom Agent** - Based on team needs

Success Criteria Template

My agent is successful when:

1. [Criterion 1]
2. [Criterion 2]
3. [Criterion 3]
4. [Criterion 4]
5. [Criterion 5]

Test scenarios: [Define 3 test cases]

Agent Definition Template

```
---
name: "agent-name"
```

```
description: 'Brief description'
tools: [changes]
model: Claude Sonnet 4
---

# Agent Name

[Role and expertise]

## Responsibilities
## Context
## Constraints
## Analysis Process
## Output Format
## Tone
```

Test, Iterate, Refine

1. **Test** with defined scenarios
2. **Observe** behavior and outputs
3. **Identify** gaps or issues
4. **Refine** instructions
5. **Re-test** until criteria met

Remember: Agents improve through iteration

Deliverables

By the end of this lab:

✅ Custom agent definition file
✅ Test results showing success
✅ Usage documentation
✅ At least one refinement iteration

Module 6

Wrap-Up & Next Steps

Governance and Continuous Improvement

Duration: 10 minutes

Discussion Questions

1. Which workflows benefit most from agents?
2. What should be standardized at team vs org level?
3. How do we prevent "prompt sprawl"?
4. How should agents be reviewed and evolved?

Best Practices for Teams

Start Small

- Begin with **reviewer agents**, not executors
- Validate value before scaling

Maintain a Catalog

- Centralized registry of agents
- Clear usage guidelines

Govern as Assets

- Version control
- PR review process
- Regular updates

Keep Humans Accountable

> Agents advise, humans decide

- Never blindly trust agent output
- Validate recommendations
- Use agents as **first pass**, not final word
- Maintain human oversight

Taking Agents to Production

Before sharing:

1. Test with real scenarios
2. Get peer review
3. Document edge cases
4. Add to team catalog
5. Set up governance

Continuous improvement:

- Collect usage feedback
- Track common issues
- Update based on lessons
- Retire obsolete agents

Adoption Roadmap

Week 1: Use existing agents in daily work
Week 2: Identify repetitive workflow → draft agent
Week 3: Test and refine with real scenarios
Week 4: Share with team and gather feedback

Key Takeaways

✅ **Ask, Edit, Agent** - Use the right mode
✅ **Custom agents** - Specialists, not prompts
✅ **Role-based design** - Focus on WHO
✅ **Iterate continuously** - Agents improve over time
✅ **Govern as assets** - Version, review, maintain
✅ **Humans accountable** - Agents assist, you decide

Resources

📚 Documentation

- Custom Agent Catalog
- Agent Design Guide
- Agent Governance

🔗 Labs

- All labs in `docs/labs/`
- Agent definitions in `.github/agents/`

Thank You!

Questions?

Feedback and contributions:
CONTRIBUTING.md

Workshop repository:
GitHub: ai-code-workshop

Next Steps

1. **Practice** with existing agents
2. **Build** your own agent
3. **Share** with your team
4. **Iterate** based on feedback
5. **Govern** as team assets

Remember: Agents are products, not prompts

Contact & Resources

Workshop Materials:

- Slides: `docs/presentations/`
- Labs: `docs/labs/`
- Agents: `.github/agents/`

GitHub Copilot Documentation:

- Custom Agents Guide
- VS Code Extension

Questions? Open an issue or contact facilitators