Day 2: Intermediate Application with Archive Integration

GitHub Spec Kit Intensive Training - Intermediate Day

Duration: 6-8 hours

Objective: Apply SDD to real HX-Infrastructure scenarios, master archive integration, develop

intermediate proficiency

Success Criteria: Complete HX-Infrastructure project using Spec Kit, demonstrate 70-80%

autonomous proficiency

(iii) HX-Infrastructure Integration: Content organization, sprint documentation, and operational runbook creation



Morning Session (3-4 hours)

Hour 1: Day 1 Review & HX-Infrastructure Analysis

1.1 Foundation Skills Validation (30 minutes)

Quick Proficiency Check:

```bash

# Verify yesterday's work is accessible

cd /home/ubuntu/github\_spec\_training/HX-Infrastructure-Knowledge-Base ls -la docs/

# Validate Day 1 deliverables

cat docs/analysis/day1-assessment.md | head -20 cat docs/adrs/ADR-0001-training-integration.md | head -20 ```yaml

### **Review Checklist:**

- [ ] Day 1 analysis completed and documented
- -[] ADR-0001 created with integration approach
- [ ] Implementation plan exists and is detailed
- [ ] Stakeholder analysis covers all groups

### 1.2 HX-Infrastructure Archive Content Analysis (30 minutes)

### **Archive Integration Planning:**

Based on the integration plan, we'll focus on high-priority content:

- 1. **Sprint Summaries & Lessons** → docs/history/sprints/
- 2. **Architecture Documentation** → docs/architecture/
- 3. **Best Practices & Runbooks** → docs/operations/
- 4. Configuration Templates → templates/
- 5. **Key Decision Records** → docs/adrs/

#### @ Practical Exercise:

Create specifications for each content category:

```bash

cd HX-Infrastructure-Knowledge-Base

Create content integration specifications

uvx -from git+https://github.com/github/spec-kit.git specify init sprint_documentation -ai copilot uvx -from git+https://github.com/github/spec-kit.git specify init architecture_docs -ai copilot uvx -from git+https://github.com/github/spec-kit.git specify init operations_runbooks -ai copilot ```yaml

Hour 2: Sprint Documentation Integration

2.1 Sprint Summary Creation (45 minutes)

Sprint Documentation Structure:

Based on the integration plan, create comprehensive sprint summaries:

Sprint 1: Repository Restructuring & CI/CD

- Objective: Transform basic infrastructure into structured, automated system
- Key Achievements: Repository organization, initial CI/CD pipeline, architecture documentation
- Lessons Learned: Importance of early structure, automation benefits, documentation discipline
- Challenges: Over-scoping, branch management, initial complexity

(6) Implementation Task:

```bash

# **Create sprint documentation structure**

mkdir -p docs/history/sprints

mkdir -p docs/architecture

mkdir -p docs/operations/runbooks

```yaml

Create detailed sprint summaries using Spec Kit:

- 1. Use /specify to define sprint documentation requirements
- 2. Use /plan to break down content creation tasks
- 3. Use /implement to create structured documentation
- 4. Use /validate to ensure completeness and quality

Sprint Documentation Template:

```markdown

# **Sprint [N]: [Theme]**

### **Overview**

Duration: [Timeframe] Objective: [Primary goal]

• Success Criteria: [Measurable outcomes]

# **Key Achievements**

• [Achievement 1 with metrics]

• [Achievement 2 with metrics]

• [Achievement 3 with metrics]

# **Technical Implementation**

### **Architecture Changes**

• [Change 1 with rationale]

• [Change 2 with rationale]

# **Automation Improvements**

• [Improvement 1 with impact]

• [Improvement 2 with impact]

### **Lessons Learned**

### What Worked Well

- [Success 1 with explanation]
- [Success 2 with explanation]

### What Could Be Improved

• [Challenge 1 with solution]

• [Challenge 2 with solution]

# **Metrics and Impact**

| Metric     | Before  | After   | Improvement |
|------------|---------|---------|-------------|
| [Metric 1] | [Value] | [Value] | [%]         |
| [Metric 2] | [Value] | [Value] | [%]         |

# **Next Sprint Preparation**

- [Preparation item 1]
- [Preparation item 2]
  - ```yaml

### 2.2 Architecture Documentation Integration (30 minutes)

#### **Architecture Documentation Focus:**

- 1. System Overview: High-level architecture and component relationships
- 2. Multi-Cloud Strategy: AWS, Azure, GCP integration approach
- 3. Al Integration Patterns: How Al components fit into infrastructure
- 4. Security Architecture: Security controls and compliance measures

### **(6)** Implementation:

Create docs/architecture/overview.md with:

- System architecture diagrams (using Mermaid)
- Component interaction patterns
- Technology stack decisions
- Scalability considerations

### **Hour 3: Operational Runbook Development**

### 3.1 Runbook Creation Strategy (45 minutes)

#### **Runbook Categories:**

- 1. **Deployment Procedures:** Step-by-step deployment guides
- 2. Monitoring & Alerting: System monitoring and incident response
- 3. Backup & Recovery: Data protection and disaster recovery
- 4. Troubleshooting: Common issues and resolution procedures

#### @ Runbook Development:

Using Spec Kit methodology:

### 1. Specify Runbook Requirements:

- Clear step-by-step procedures
- Prerequisites and assumptions
- Success criteria and validation steps
- Error handling and rollback procedures

#### 2. Plan Runbook Structure:

- Logical flow and sequencing
- Decision points and branching
- Resource requirements and dependencies
- Time estimates and complexity levels

### 3. Implement Runbook Content:

- Detailed procedural steps
- Code examples and commands
- Screenshots and diagrams where helpful
- Validation checkpoints

#### 4. Validate Runbook Effectiveness:

- Test procedures in safe environment
- Review with subject matter experts

- Gather feedback from potential users
- Iterate based on testing results

#### **Sample Runbook Structure:**

```markdown

[Procedure Name] Runbook

Overview

• Purpose: [What this procedure accomplishes]

• Scope: [When to use this procedure]

• Prerequisites: [Required access, tools, knowledge]

• Estimated Time: [Duration for completion]

Pre-Procedure Checklist

• [] [Prerequisite 1]

• [] [Prerequisite 2]

• [] [Prerequisite 3]

Procedure Steps

Step 1: [Step Name]

Objective: [What this step accomplishes]

Commands:

bash

[command 1]

[command 2] yaml

Expected Output: [What you should see] **Validation:** [How to confirm success]

Step 2: [Step Name]

[Continue pattern...]

Troubleshooting

| Issue | Symptoms | Resolution |
|-----------|------------|------------|
| [Issue 1] | [Symptoms] | [Solution] |
| [Issue 2] | [Symptoms] | [Solution] |

Rollback Procedure

[Steps to undo changes if needed]

Post-Procedure Validation

- [] [Validation check 1]
- [] [Validation check 2]
- [] [Validation check 3]
 - ```yaml

3.2 Monitoring and Incident Response (30 minutes)

Monitoring Framework Integration:

Create comprehensive monitoring documentation based on Sprint 2 learnings:

1. Monitoring Strategy:

- Key performance indicators (KPIs)
- Alert thresholds and escalation procedures
- Dashboard design and access controls
- Synthetic monitoring and health checks

2. Incident Response Procedures:

- Incident classification and severity levels
- Response team roles and responsibilities
- Communication protocols and stakeholder updates
- Post-incident review and improvement processes

@ Implementation Task:

Create docs/operations/runbooks/incident-response.md with:

- Incident response workflow
- Escalation matrix
- Communication templates
- Post-mortem process

🌇 Afternoon Session (3-4 hours)

Hour 4: Advanced Specification Patterns

4.1 Complex System Specifications (45 minutes)

Multi-Component System Specification:

Learn to specify complex systems with multiple interacting components:

1. System Boundary Definition:

- Clear scope and interface definitions
- External dependencies and integrations
- Data flow and communication patterns
- Security and compliance requirements

2. Component Interaction Specifications:

- API contracts and data formats
- Error handling and retry logic
- Performance requirements and SLAs
- Monitoring and observability needs

@ HX-Infrastructure System Specification:

Create a comprehensive specification for the HX-Infrastructure system:

```bash

# **Create system specification**

uvx -from git+https://github.com/github/spec-kit.git specify init hx\_infrastructure\_system -ai copilot
```yaml

Specification Components:

- 1. Infrastructure Layer: Compute, storage, networking
- 2. Platform Layer: Container orchestration, service mesh
- 3. **Application Layer:** Microservices, APIs, databases
- 4. **Operations Layer:** Monitoring, logging, security
- 5. Al Layer: ML pipelines, model serving, automation

4.2 Configuration Management Specifications (30 minutes)

Configuration as Code Approach:

Specify configuration management using Infrastructure as Code principles:

1. Environment Specifications:

- Development, staging, production configurations
- Environment-specific variables and secrets
- Deployment pipeline configurations
- Rollback and disaster recovery procedures

2. Security Configuration:

- Access controls and authentication
- Network security and firewall rules
- Encryption and key management
- Compliance and audit requirements

@ Configuration Templates:

Create reusable configuration templates in templates/ directory:

- Ansible playbook templates
- Terraform module templates
- CI/CD pipeline templates
- Monitoring configuration templates

Hour 5: Quality Assurance and Testing

5.1 Specification Testing Strategies (45 minutes)

Testing Pyramid for Specifications:

- 1. Unit Level: Individual component specifications
- 2. Integration Level: Component interaction specifications
- 3. System Level: End-to-end system specifications
- 4. Acceptance Level: User journey and business value specifications

@ HX-KB Testing Implementation:

Create testing strategies for knowledge base content:

1. Content Validation Tests:

- Link checking and reference validation
- Code example testing and verification
- Documentation completeness checks
- Format and style consistency validation

2. Workflow Testing:

- Contribution process validation
- Review and approval workflow testing
- Automated content generation testing
- Search and navigation functionality testing

Testing Implementation:

```bash

# **Create testing framework**

mkdir -p tests/content mkdir -p tests/workflows mkdir -p .github/workflows

# Create content validation workflow

cat > .github/workflows/content-validation.yml << 'EOF'

name: Content Validation

on: push:

branches: [ main, develop ]

pull\_request: branches: [ main ]

iobs:

validate-content:

runs-on: ubuntu-latest

permissions: contents: read pull-requests: read

steps:

- uses: actions/checkout@b4ffde65f46336ab88eb53be808477a3936bae11 # v4

```
- name: Validate Markdown Links
 uses: gaurav-nelson/github-action-markdown-link-check@v1
 with:
 use-quiet-mode: 'yes'
 use-verbose-mode: 'yes'
 config-file: '.github/workflows/markdown-link-check-config.json'
- name: Validate Documentation Structure
 run:
 # Check required directories exist
 test -d docs/adrs || exit 1
 test -d docs/architecture || exit 1
 test -d docs/operations || exit 1
 # Check required files exist
 test -f README.md || exit 1
 test -f docs/adrs/ADR-0001-training-integration.md \mid\mid exit 1
 echo "Documentation structure validation passed"
- name: Validate Code Examples
 run: |
 # Extract and test bash code blocks
 find docs -name "*.md" -exec grep -l "```bash" {} \\; | while read file; do
 echo "Validating bash examples in $file"
 # Add bash syntax validation here
 done
```

EOF ```yaml

#### 5.2 Continuous Integration for Documentation (30 minutes)

### **Documentation CI/CD Pipeline:**

Implement automated validation and deployment for knowledge base content:

### 1. Content Quality Gates:

- Markdown linting and formatting
- Spell checking and grammar validation
- Link checking and reference validation
- Code example testing

#### 2. Automated Content Generation:

- Table of contents generation
- Cross-reference index creation
- Metrics dashboard updates
- Search index maintenance

### **©** CI/CD Implementation:

Enhance the existing connectivity check workflow with comprehensive validation:

```
```yaml
name: Enhanced Content Validation
on:
push:
branches: [ main ]
```

```
pull_request:
branches: [ main ]
schedule:
- cron: '0 2 * * 1' # Weekly validation

jobs:
content-validation:
runs-on: ubuntu-latest
permissions:
contents: read
pull-requests: read
steps:
- name: Checkout repository
uses: actions/checkout@b4ffde65f46336ab88eb53be808477a3936bae11 # v4
```

```
- name: Setup Node.js
 uses: actions/setup-node@60edb5dd545a775178f52524783378180af0d1f8 # v4
 with:
    node-version: 18
- name: Install validation tools
  run: |
    npm install -g markdownlint-cli
    npm install -g markdown-link-check
- name: Validate markdown format
 run: markdownlint docs/ --config .markdownlint.json
- name: Check internal links
  run: find docs -name "*.md" -exec markdown-link-check {} \\;
- name: Validate documentation completeness
  run:
    if [ -f scripts/validate-completeness.py ]; then python3 scripts/validate-com-
pleteness.py; else echo "Skipping completeness validation - script not found"; fi
- name: Generate content metrics
    if [ -f scripts/generate-metrics.py ]; then python3 scripts/generate-metrics.py >
metrics/content-metrics.json; else echo "Skipping metrics generation - script not
found"; fi
- name: Update search index
  run: |
   if [ -f scripts/update-search-index.py ]; then python3 scripts/update-search-in-
dex.py; else echo "Skipping search index update - script not found"; fi
```

```yaml

### **Hour 6: Integration and Deployment**

### **6.1 Knowledge Base Content Integration (45 minutes)**

### **Content Population Strategy:**

Systematically populate the knowledge base with integrated content:

#### 1. Sprint Documentation Integration:

- Create all four sprint summary documents

- Include lessons learned and best practices
- Add metrics and performance data
- Link to relevant architecture decisions

#### 2. Architecture Documentation:

- System overview and component diagrams
- Technology stack decisions and rationale
- Security architecture and compliance
- Scalability and performance considerations

#### 3. Operational Procedures:

- Deployment runbooks and procedures
- Monitoring and alerting configurations
- Incident response and troubleshooting guides
- Backup and recovery procedures

### **©** Content Creation Tasks:

```bash

Create comprehensive content structure

mkdir -p docs/history/sprints

mkdir -p docs/architecture/diagrams

mkdir -p docs/operations/{runbooks,monitoring,backup-recovery}

mkdir -p docs/security

mkdir -p docs/troubleshooting

mkdir -p templates/{ansible,terraform,cicd}

mkdir -p examples/{playbooks,modules,configs}

Populate with structured content

Use Spec Kit to generate content systematically

```yaml

### 6.2 Validation and Quality Assurance (30 minutes)

### **Comprehensive Quality Review:**

Validate all integrated content for quality and completeness:

### 1. Content Quality Checklist:

- [ ] All placeholder sections populated
- [ ] Consistent formatting and style
- [ ] Working internal and external links
- [ ] Accurate and tested code examples
- [ ] Complete cross-references and navigation

#### 2. Functional Validation:

- -[] Search functionality working
- [ ] Navigation paths clear and logical
- -[] Contribution workflow tested
- -[] Automated validation passing

#### 3. Stakeholder Review:

- [ ] End user feedback collected
- -[] Contributor workflow validated
- [ ] Maintainer procedures tested
- [ ] Training integration confirmed

### @ Quality Assurance Implementation:

Run comprehensive validation suite:

```bash

Run all validation checks

.github/workflows/content-validation.yml

Test contribution workflow

git checkout -b test-contribution echo "Test content" > docs/test.md git add docs/test.md git commit -m "Test contribution workflow" git push origin test-contribution

Create test PR and validate review process

```yaml

### 6.3 Day 2 Deliverables and Wrap-up (15 minutes)

### **Day 2 Completion Checklist:**

- [ ] Sprint documentation completed for all 4 sprints
- [ ] Architecture documentation created and validated
- [ ] Operational runbooks developed and tested
- [ ] Configuration templates created and documented
- [ ] Quality assurance processes implemented
- [ ] Content validation workflows established

### **Success Metrics:**

- Content completeness: 80%+ of planned content integrated
- Quality score: 85%+ on validation checks
- Workflow functionality: All automated processes working
- Stakeholder satisfaction: Positive feedback on usability

### @ Day 2 Deliverables:

Commit and push the following to HX-KB repository:

- docs/history/sprints/sprint-[1-4]-summary.md Complete sprint documentation
- docs/architecture/overview.md System architecture documentation
- docs/operations/runbooks/ Operational procedures and runbooks
- templates/ Configuration and deployment templates
- .qithub/workflows/content-validation.yml Enhanced validation workflow
- docs/adrs/ADR-0002-content-integration.md Content integration decisions



### 📚 Resources and References

### **Day 2 Specific Resources**

- Sprint Documentation Template (../templates/sprint-summary-template.md)
- Architecture Documentation Guide (../docs/guides/architecture-documentation.md)
- Runbook Development Best Practices (../docs/guides/runbook-best-practices.md)
- Content Validation Tools (tools/content-validation/)

### Integration References

- HX-Infrastructure Integration Plan (docs/IntegrationPlan.md)
- Archive Content Mapping (docs/integration/content-mapping.md)
- Quality Assurance Framework (docs/qa/framework.md)



## @ Day 2 Success Criteria

#### **Technical Proficiency:**

- -[] 70-80% autonomous proficiency with Spec Kit
- [ ] Complex system specifications created and validated
- [ ] Quality assurance processes implemented
- [ ] Content integration completed successfully

#### **Project Outcomes:**

- [ ] Comprehensive sprint documentation created
- [ ] Architecture documentation completed
- [ ] Operational runbooks developed and tested
- [ ] Configuration templates created and validated

### **Knowledge Retention:**

- [ ] Advanced specification patterns mastered
- [ ] Quality assurance principles applied
- [ ] Integration strategies understood
- [ ] Stakeholder needs addressed

### **Next Day Preparation:**

- [ ] Day 3 objectives reviewed
- [ ] Advanced optimization topics prepared
- [ ] Complex scenario materials ready
- -[] Team coordination confirmed

 $\leftarrow$  Day 1: Foundation (day1\_foundation.md) | Day 3: Advanced Techniques (day3\_advanced.md)  $\rightarrow$