Reviewing and Testing Code Using Cursor and Claude

AI as Your Quality Partner →

Contact Info

Ken Kousen • Kousen IT, Inc.

- ken.kousen@kousenit.com
- http://www.kousenit.com
- http://kousenit.org (blog)

Connect with Me

- **Twitter:** @kenkousen
- **Bluesky:** @kousenit.com
- **LinkedIn:** linkedin.com/in/kenkousen
- **Newsletter:** kenkousen.substack.com
- **YouTube:** @talesfromthejarside

Course Overview: 5 Sessions

1. **Using Cursor for Java Development** (Session 1 - Complete)
   - Understanding code, navigation, generation, testing
2. **Using Cursor for Mobile Development** (Session 2 - Complete)
   - Android/Kotlin with AI assistance
3. **Agentic Coding with Cursor** (Session 3 - Complete)
   - AI as collaborative partner for complex projects
4. **Reviewing and Testing Code** (Today - 3 hours)
   - AI as quality assurance partner
5. **Exploring Agents and MCP** (3 hours)
   - Model Context Protocol and advanced features

Today’s Session: What We’ll Cover

- **AI-Assisted Test Generation** - Comprehensive test suites with AI
- **Comprehensive Unit Testing** - JUnit 5, Mockito, and AssertJ
- **Integration Testing Strategies** - TestContainers and service mocking
- **End-to-End Testing** - Complete workflow validation
- **AI-Powered Debugging** - Intelligent troubleshooting

Project 1: E-Commerce Testing Suite

Build comprehensive testing strategy with AI assistance:

- Unit Tests (JUnit 5, Mockito, AssertJ)
- Integration Tests (TestContainers, @DataJpaTest)
- End-to-End Tests (API testing, workflow validation)

Project 2: Legacy Code Testing

Apply AI testing strategies to real-world legacy systems

- Analyze existing code without tests
- Create testing strategies incrementally
- Refactor for testability

Sessions 1-3 Recap

- **Chat Mode** (Cmd/Ctrl+L) - Ask questions, understand code
- **Agent Mode** (Cmd/Ctrl+I) - Generate code, refactor
- **Composer Mode** (Cmd/Ctrl+Shift+I) - Multi-file generation
- **Extended Thinking** - Complex analysis and planning

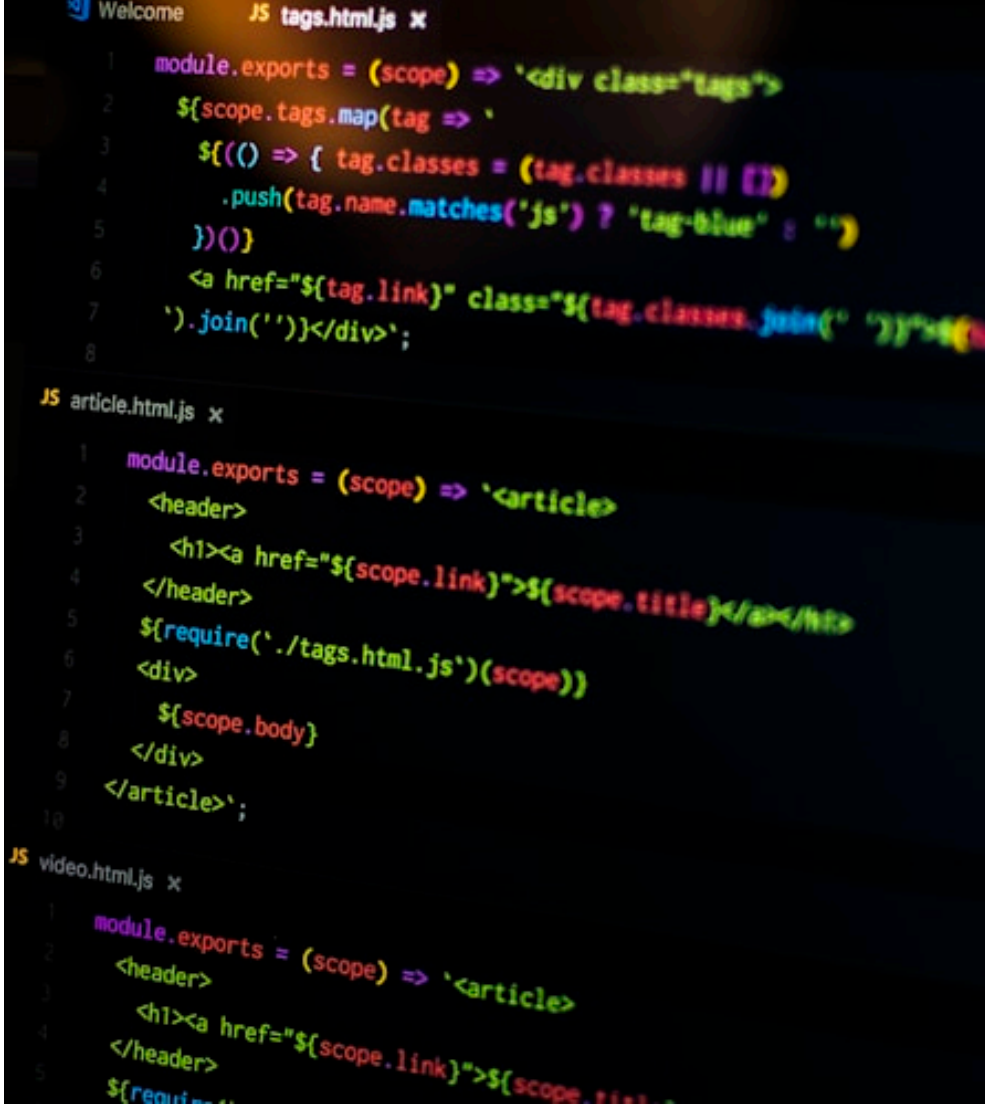
Today: AI Testing

- **Test Generation** - AI creates comprehensive test suites
- **Quality Analysis** - AI reviews code for issues
- **Integration Testing** - TestContainers and mocking
- **Debugging Assistance** - AI troubleshoots failures

Part 1: AI-Assisted Test Generation

The Testing Paradigm Shift

- From manual to AI-assisted
- From reactive to proactive
- From limited to comprehensive



Traditional Testing

- Manual test writing
- Limited test coverage
- Reactive quality assurance
- Time-consuming maintenance

AI-Assisted Testing

- Automated test generation
- Comprehensive coverage analysis
- Proactive quality assurance
- Intelligent test maintenance

AI Testing Philosophy

1. Coverage-Driven

AI identifies gaps: edge cases, untested paths, boundary values

2. Scenario-Based

AI generates realistic test scenarios and user journeys

AI Testing Philosophy (continued)

3. Maintenance-Aware

AI helps keep tests up-to-date with code changes

4. Quality-Focused

AI ensures meaningful assertions and realistic test data

Unit Test Generation

- Method-level testing with edge cases
- Mocking strategies
- Assertion patterns
- Error condition testing

Integration Test Generation

- Component interaction testing
- Database integration
- External service mocking
- Transaction testing

End-to-End Test Generation

- Full workflow testing
- User journey validation
- API contract testing
- Cross-system integration

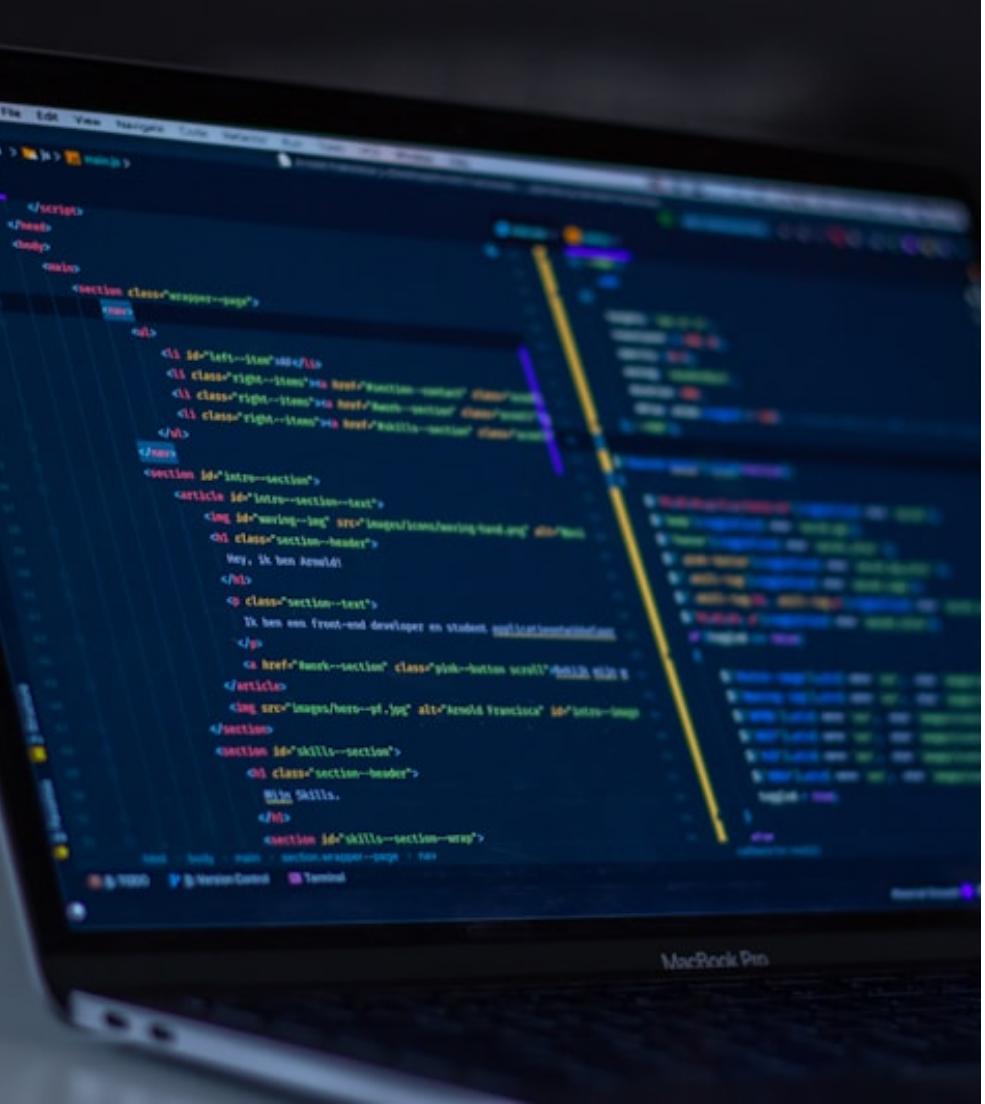
Demo: AI Test Generation

Service Layer Testing:

```
Extended Thinking: "Generate comprehensive unit tests for UserService.
Include happy path, edge cases, and error conditions.
Use JUnit 5, Mockito, and AssertJ."
```

AI Response Process

1. **Analysis** - Understanding the service
2. **Strategy** - Identifying test scenarios
3. **Generation** - Creating tests
4. **Validation** - Ensuring quality



Part 2: Comprehensive Unit Testing

JUnit 5, Mockito, and AssertJ

- Modern testing stack
- AI-assisted generation
- Quality-focused approach

Spring Boot 3.4+ Testing Changes

Important Update:

`@MockBean` and `@SpyBean` are deprecated as of Spring Boot 3.4

New annotations:

- `@MockitoBean` - replaces `@MockBean`
- `@MockitoSpyBean` - replaces `@SpyBean`

New Package Location

```
import org.springframework.test.context.bean.override.mockito.MockitoBean;
import org.springframework.test.context.bean.override.mockito.MockitoSpyBean;
```

Migration: AI can help update your tests automatically!

Unit Testing Strategies

Service Layer

Mock dependencies, test business logic, verify error handling

Repository Layer

@DataJpaTest for CRUD operations and custom queries

Unit Testing Strategies (continued)

Controller Layer

@WebMvcTest for endpoints, validation, and responses

Utilities

Test static methods, edge cases, and error conditions

Demo: Service Layer Testing

Prompt:

```
"Generate unit tests for UserService using JUnit 5,
Mockito, and AssertJ."
```

Generated Test Structure

```
@ExtendWith(MockitoExtension.class)
class UserServiceTest {
    @Mock private UserRepository repo;
    @InjectMocks private UserService service;

    @Test
    void shouldCreateUserWithValidData() {
        when(repo.save(any())).thenReturn(savedUser);
        User result = service.createUser(request);
        assertThat(result).isNotNull();
        verify(repo).save(any());
    }
}
```

Demo: Repository Testing

Prompt:

```
"Create unit tests for UserRepository using @DataJpaTest.
Test all CRUD operations and custom query methods."
```

Generated Repository Test

```
@DataJpaTest
class UserRepositoryTest {
    @Autowired
    private TestEntityManager entityManager;
    @Autowired
    private UserRepository userRepository;

    @Test
    void shouldFindUserByEmail() {
        User user = new User("John", "Doe", "john@example.com");
        entityManager.persistAndFlush(user);

        Optional<User> result =
            userRepository.findByEmail("john@example.com");

        assertThat(result).isPresent();
    }
}
```

Student Exercise: Controller Testing

Your Turn:

Generate tests using this prompt:

```
"Generate unit tests for UserController using @WebMvcTest.
Use @MockitoBean for service mocking (Spring Boot 3.4+).
Test endpoints, validation, and error handling."
```

Controller Testing Goals

- **Review:** Endpoint coverage, MockMvc usage, JSON assertions
- **Refine:** Add edge cases and improve test data
- **Note:** Spring Boot 3.4+ uses `@MockitoBean`

Part 2B: AI-Powered Test Quality

Review and Improvement

Demo: AI Test Quality Review

Prompt:

```
"Review these unit tests and suggest improvements.
Focus on test coverage, assertion quality, and maintainability."
```

AI Test Review Insights

- **Missing test cases** - Edge cases, error conditions
- **Weak assertions** - Too generic, not specific enough
- **Test smells** - Unclear names, too much setup
- **Improvement suggestions** - Better structure, clearer intent

Demo: Test Data Generation

Prompt:

```
"Generate realistic test data for User and Order entities.
Include various scenarios: valid data, edge cases, invalid data."
```

AI-Generated Test Data: Valid Users

```
// Valid users
User normalUser = new User("John", "Doe", "john@example.com");
User longNameUser = new User("Christopher", "Bartholomew",
    "christopher.bartholomew@example.com");
```

AI-Generated Test Data: Edge Cases

```
// Edge cases
User shortNameUser = new User("A", "B", "a@b.co");
User unicodeUser = new User("José", "García", "jose@example.es");

// Invalid data for validation testing
User nullEmailUser = new User("John", "Doe", null);
User invalidEmailUser = new User("John", "Doe", "not-an-email");
```

Student Exercise: Improve Your Tests

Review Generated Tests:

```
"Review my UserService tests. What's missing?
What could be improved?"
```

Student Exercise: Generate Test Data

Generate Test Data:

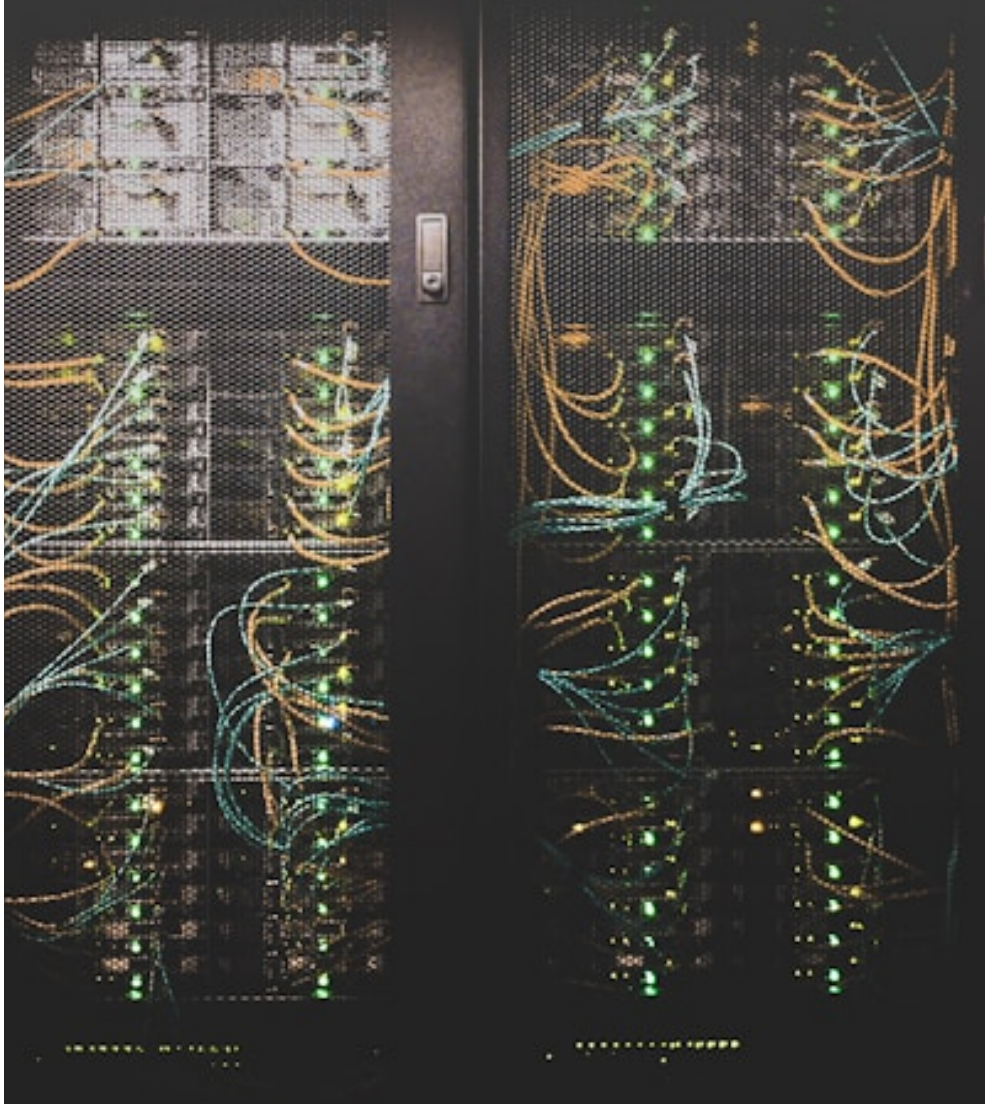
```
"Generate test data for various Order scenarios:
new orders, completed orders, cancelled orders."
```

Goal: Write higher quality tests with AI assistance

Part 3: Integration Testing

TestContainers and Service Mocking

- Real database testing
- Component interaction
- External service mocking



Database Integration Testing

TestContainers for real databases, SQL queries, transactions

Service Integration Testing

Test component interactions, data flow, error propagation

External Service Testing

WireMock for HTTP services, payment gateways, third-party APIs

Demo: TestContainers Integration

Prompt:

```
"Set up TestContainers with PostgreSQL for integration testing"
```

Generated TestContainers Setup

```
@SpringBootTest
@Testcontainers
class UserIntegrationTest {
    @Container
    static PostgreSQLContainer<?> postgres =
        new PostgreSQLContainer<>("postgres:15");

    @DynamicPropertySource
    static void props(DynamicPropertyRegistry r) {
        r.add("spring.datasource.url", postgres::getJdbcUrl);
    }

    @Test
    void shouldCreateAndRetrieveUser() { /* ... */ }
}
```

Demo: Service Integration Testing

Complete Module Testing:

```
Plan Mode: "Create integration tests for the complete User module
including service, repository, and database layers."
```

Generated Integration Tests

- Service layer integration
- Repository integration
- Database transaction testing
- Error handling verification

Integration Test Benefits

- Real database testing
- Actual component interaction
- Transaction boundary testing
- Error scenario testing

Student Exercise: External Service Mocking

Prompt:

```
"Set up WireMock for mocking external services.
Create integration tests for payment processing."
```

WireMock Configuration Goals

- Payment gateway mocking
- Email service mocking
- Error scenario testing
- Response validation

Break Time!

10 Minutes

Stretch, grab coffee, be back on time!

Part 4: End-to-End Testing

Complete Workflow Validation

- Full user journeys
- API contracts
- Cross-system integration

End-to-End Testing Strategies

- **API Testing:** Complete workflows, authentication
- **Workflow Testing:** User journeys, business processes
- **Cross-System Testing:** Multi-service integration

Demo: Complete API Testing

Prompt:

```
"Create comprehensive API tests for the e-commerce application.
Include authentication, authorization, and error handling."
```

Generated API Test Coverage

- Authentication flow testing
- Authorization scenario testing
- CRUD operation validation
- Error response verification

API Test Structure

```
@SpringBootTest(webEnvironment = RANDOM_PORT)
class EcommerceApiIntegrationTest {

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void shouldCompleteFullOrderWorkflow() {
        // Test implementation
    }
}
```

Student Exercise: Workflow Testing

Create Workflow Tests:

```
Plan Mode: "Create end-to-end tests for the complete order processing
workflow from user login to order completion."
```

Workflow Test Scenarios

- User registration and login
- Product browsing and selection
- Cart management
- Checkout process

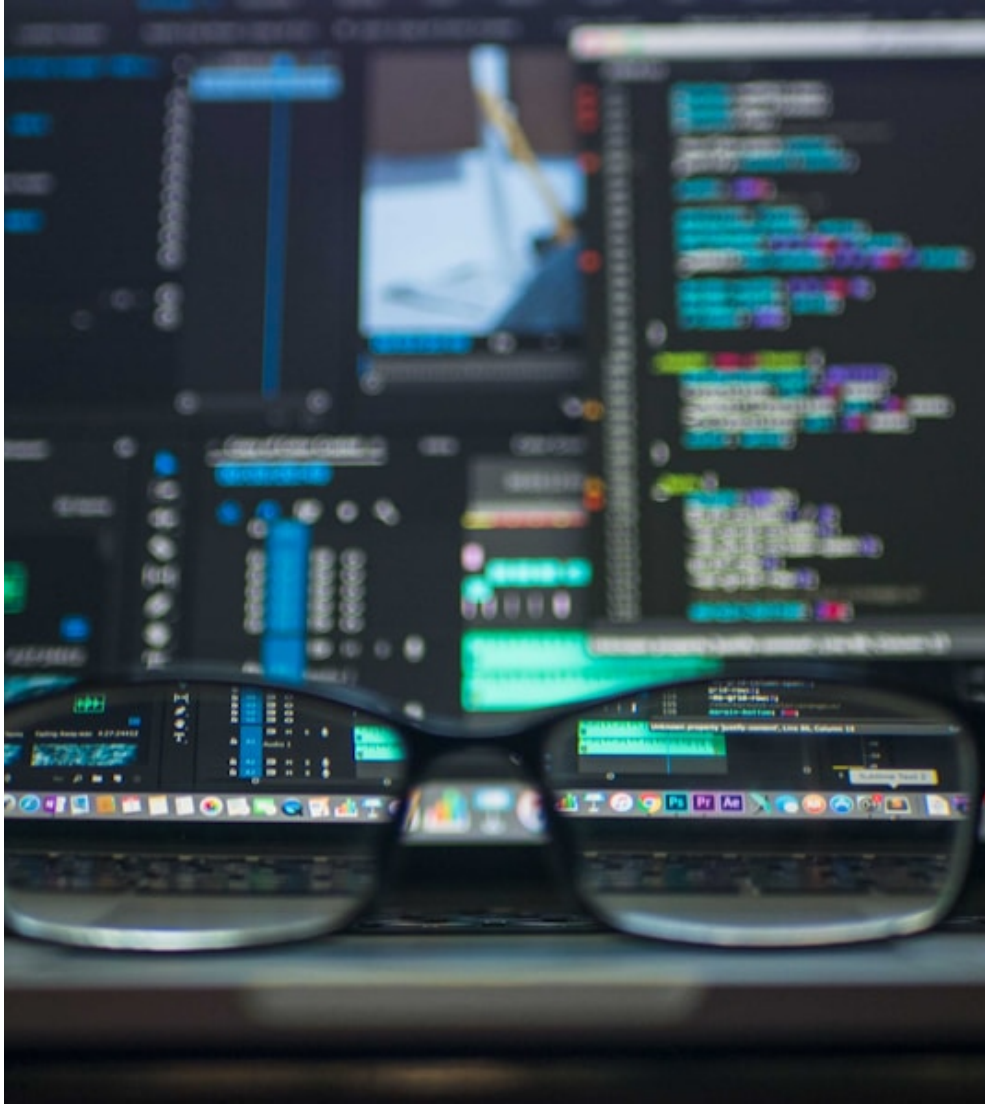
Workflow Test Implementation

- Test data setup
- Workflow execution
- State validation
- Response verification

Part 5: AI-Powered Debugging

Intelligent Troubleshooting

- Error analysis
- Root cause identification
- Fix recommendations



Error Analysis with AI

Stack traces, root causes, error patterns

Test Failure Debugging

Failures, environment issues, data problems, configuration

Demo: AI-Guided Debugging

Prompt:

```
"Analyze this failing test and provide debugging guidance.
Identify the root cause and suggest fixes."
```

AI Debugging Process

1. **Error Analysis** - Understanding the failure
2. **Root Cause Identification** - Finding the source
3. **Fix Recommendation** - Suggesting solutions
4. **Prevention Strategy** - Avoiding future issues

Student Exercise: Test Failure Analysis

Analyze Test Failure:

```
Chat Mode: "This test is failing with a NullPointerException.
Help me debug the issue and fix the test."
```

Debug Process Steps

- Analyze error message
- Identify root cause
- Implement fix
- Validate solution

Part 6: Legacy Code Testing

Testing Strategies for Legacy Systems

- Incremental approach
- Risk-based testing
- Modernization path

Legacy Code Testing Challenges

- Limited test coverage
- Tightly coupled code
- Missing documentation
- Complex business logic

Legacy Code Testing Strategies

- Incremental test addition
- Characterization testing
- Refactoring for testability
- Risk-based testing

Demo: Legacy Code Analysis

Legacy Testing Strategy:

```
Extended Thinking: "Analyze this legacy Java application and create
a testing strategy. Identify testing challenges and recommend
approaches for improving test coverage."
```

Analysis Results

- Current test coverage assessment
- Testing challenge identification
- Risk-based testing approach
- Modernization recommendations

Recommended Approach

1. **Characterization Testing** - Understand current behavior
2. **Incremental Testing** - Add tests gradually
3. **Refactoring** - Improve code structure
4. **Comprehensive Testing** - Full coverage

Student Exploration Exercise

Use Extended Thinking to explore:

- "What are the main testing challenges in this legacy codebase?"
- "How would you prioritize testing improvements?"
- "What risks should be considered?"

Wrap-Up

Key Takeaways

AI Testing Decision Tree

```
Need unit tests? → Use Agent Mode with JUnit 5
Need integration tests? → Use TestContainers with Plan Mode
Need end-to-end tests? → Use Extended Thinking for workflow design
Need debugging help? → Use Chat Mode for analysis
```

What We Accomplished Today

- ✅ **Test Suite Generation** using AI
- ✅ **Unit Testing** with JUnit 5, Mockito, AssertJ
- ✅ **Integration Testing** with TestContainers

What We Accomplished Today (continued)

- ✅ **End-to-End Testing** for workflows
- ✅ **AI-Powered Debugging**
- ✅ **Legacy Code Testing** strategies

Best Practices: Generation

- Start with AI test generation for comprehensive coverage
- Use appropriate testing strategies (unit, integration, E2E)
- Master the stack: JUnit 5, Mockito, AssertJ, TestContainers

Best Practices: Quality

- Leverage AI for debugging and troubleshooting
- Apply testing to legacy code incrementally
- Use realistic test data from AI generation

Comparing Sessions 1-4

Sessions 1-3	Session 4
Code generation	Test generation
Architecture planning	Testing strategy
AI collaboration	AI quality partnership

Same Core Principles

- Natural language as interface
- Iterative refinement
- Context awareness
- Human-AI collaboration

Preview: Session 5

Building AI-Powered Java Apps with Spring AI

- ChatClient and prompt templates
- RAG (Retrieval Augmented Generation)
- Function calling and tools
- Model Context Protocol (MCP)

Lab Exercises

Part A: Reinforce Today's Learning

- Complete testing suite generation
- Apply AI debugging to test failures
- Create custom testing workflows

Lab Exercises (continued)

Part B: Legacy Code Testing

- Analyze provided legacy codebase
- Create testing strategy
- Apply incremental testing approach

See `labs.md` for details

Resources: Testing

- JUnit 5 User Guide
- TestContainers Documentation
- Mockito Documentation
- AssertJ Documentation

Resources: Spring

- Spring Boot Testing
- Cursor Documentation

Questions?

AI test generation • Unit testing • Integration testing

End-to-end testing • AI debugging • Legacy code

Thank You!

ken.kousen@kousenit.com • kousenit.com • @kenkousen

Apply AI testing to your projects

Join us for Session 5!