

M07 — Testing, QA & Validation with AI Lab

Lab — Generative AI Boot Camp for Developers — Exported 2026-08-03

Time: ~25 min | **Day 2 PM** (shared with M06)

2026-08-03

Learning Objectives

Generate and evaluate AI-produced test suites. Learn when to trust AI-generated tests and when human judgment is essential.

2026-08-03

Concept Review: AI-Assisted Testing

What AI Excels At in Testing

Task	AI Capability	Human Role
Unit test generation	Generates happy path + common edge cases from function signatures	Add domain-specific edge cases; verify assertions
Edge case discovery	Suggests scenarios humans often miss (null, boundary, type errors)	Filter out unrealistic scenarios
Test data generation	Creates large, varied datasets	Ensure data is realistic for your domain
Mutation testing	Introduces bugs → checks if tests catch them	Interpret results; strengthen weak tests

AI Testing Limitations

- **Doesn't know your business logic:** AI can test that a function returns *something* but not that it returns the *right* thing for your domain.
- **Can generate buggy tests:** AI tests can contain their own bugs — always review assertions.
- **Misses implicit requirements:** "This should never take more than 100ms" — AI won't know unless you tell it.

The AI Testing Workflow

1. Provide function + context → AI generates baseline tests
2. Run tests → identify failures
3. Ask AI: "What edge cases am I missing?"
4. Add your domain-specific cases → run again
5. Review all assertions for correctness
6. Commit — human-reviewed, AI-assisted

Exercise: Generate & Validate Tests

Step 1 — Generate Baseline Tests (10 min)

Use your code from M03 or M05, or the sample code from M02 (`workshop/m02/sample-code.py`).

Generate tests with **OpenCode**:

```
opencode run "Read workshop/m02/sample-code.py (or your own code). Generate pytest tests for the UserService class. Cover: happy path for create_user and get_user, invalid username format, invalid email format, short password, duplicate user, and user not found. Use descriptive test function names. Include fixtures for database setup."
```

Then try generating tests for the same code using **VS Code Chat** — select the code, open Chat, ask for tests. Compare the results.

Step 2 — Generate Edge Cases (5 min)

Ask the AI what you're missing:

```
opencode run "I have tests for UserService covering happy path and basic validation. What edge cases am I missing? Consider: Unicode usernames, extremely long inputs, SQL injection attempts, concurrent access, cache expiration, and password hashing edge cases. List the top 10 missing tests."
```

Pick 3–5 of the suggested edge cases and generate tests for them.

Step 3 — Run & Evaluate (10 min)

Run all tests:

```
pytest test_user_service.py -v
```

Evaluate: - How many passed on the first run? - Which failed — was it a code bug or a test bug? - Did the AI-generated edge cases catch real issues? - What tests would you add that the AI didn't suggest?

Deliverable

Test file + edge case list + evaluation notes.

2026-08-03

Troubleshooting

Symptom	Fix
Tests don't compile/import	AI used wrong imports — fix manually or ask AI to fix
Tests pass trivially	AI tested the easy path only — ask specifically for edge cases
Tests test wrong behavior	Review each assertion before running; AI may misunderstand the function
pytest not found	<code>pip install pytest</code>
Test file can't find source code	Ensure source is in the same directory or adjust PYTHONPATH

2026-08-03

Hints

- **The edge case step is the most valuable part** of this lab — AI often catches cases humans overlook.
- **AI tests are a starting point, not the finish line.** Every assertion needs human review.
- **Try both tools** (OpenCode + VS Code Chat) for test generation — do they produce different tests? Different quality?

2026-08-03