Lecture V — Verification and Testing with Agents

AI-Assisted Programming for PhD Researchers

Dr. Tobias Vlćek
Helmut Schmidt University — September 2026

The Trust-Then-Verify Gap

Plausible Is Not Correct

- AI code is optimized to look right, not to be right
- It sails through the eyeball test and fails on the edge case
- The °F bug from Lab 1 survived that test for months
- Reading code confirms it is plausible, not that it is correct

"Done" Is a Claim, Not a Fact

- Agents report success confidently, whether or not they earned it
- "Done!" is a claim about the work, not evidence about the work
- Your job is not to trust the claim — it is to make success checkable

. . .

Every "it works" you accept on faith is a bug you have agreed not to find yet.

Give the Agent a Check

The Highest-Leverage Practice

- Give the agent a command that answers one question: pass or fail
- Tests, a linter, a build, a diff against a known-good output
- With a check, the agent can verify itself — run, read the result, fix, repeat
- That is a self-correcting loop, and it runs without you in the middle

What Counts as a Check

Match the check to what changed:

Change type	Check
Data code	tests on a known input
A plot	look at the file
A refactor	tests still green
A script	run it on real data

Without a Check vs With

No check:

```
> add the parser
done! the parser
handles all the
formats now.
```

With a check:

```
> add the parser
test failed on ""...
fixing the empty case...
test passed.
```

The check turns a confident claim into an honest loop.

Testing in Fifteen Minutes

A Test Is Just a Function

A file named `test_*`, a function named `test_*`, an `assert`:

```
# tests/test_io.py
from pipeline.io import clean

def test_clean_is_callable():
    assert callable(clean)
```

Run the whole suite with one command:

```
uv run pytest
```

Arrange, Act, Assert

Build a tiny input, call the function, assert the result:

```
import pandas as pd
from pipeline.io import load_raw, clean

def test_na_humidity_becomes_nan(tmp_path):
    p = tmp_path / "m.csv"                          # Arrange
    p.write_text(
        "reading_id,station,timestamp,temperature,"
        "temp_unit,humidity_pct,wind_ms\n"
        "R0001,Alpha,2025-03-01 06:00,12.0,C,n/a,3.0\n"
    )
    df = clean(load_raw(p))                         # Act
    assert pd.isna(df.loc[0, "humidity_pct"])      # Assert
```

What to Test

- Test behavior and edge cases, not implementation details
- Behavior survives a rewrite; implementation details do not
- The edge cases are exactly your spec's acceptance criteria
- Your Lab 2 spec is already a test list in disguise

Tests Are Executable Specs

- Each acceptance criterion becomes one small test
- The spec stops being prose and starts being enforced
- "–99 → NaN" is no longer a promise — it is a check that runs
- A green suite is the spec, proven, every time you run it

TDD with Agents

Flip the Order

Write the test first, implement second:

- **Red**: write a failing test that pins the requirement
- **Green**: let the agent implement until the test passes
- **Refactor**: clean it up while the test stays green

Why This Works So Well with Agents

- The test pins the requirement so the agent cannot silently skip it
- "Make this test pass" is the perfect check-driven prompt
- The agent iterates against the check, not against your patience
- You review a passing suite, not a paragraph of reassurance

The Cardinal Sin

> ⚠ Warning
>
> Never let the agent edit tests to make them pass — the tests are read-only. A "fixed" test is often a deleted requirement.

Debugging with Agents

Where Skills Atrophy Fastest

- In the skill-formation RCT, the largest gap between the AI and hand-coding groups was on debugging questions [1]
- Debugging is the skill you most need to check the agent
- Hand all of it to the agent and you lose exactly that skill

Paste-and-Pray

- Paste the error, accept whatever comes back, move on
- Works often enough to feel fine, and teaches you nothing
- Falls apart the moment the bug is anything but shallow

- You end up unable to tell a real fix from a lucky guess

A Protocol Instead

Debug in five deliberate steps:

1. **Reproduce** the failure minimally
2. **Hypothesize** a cause — before touching any code
3. **Inspect** or instrument to test the hypothesis
4. **Fix** the actual cause, not the symptom
5. **Regression test** so it can never return silently

Making the Agent Follow It

Force the hypotheses out before any edit:

```
Reproduce the failure first. Then give me
three ranked hypotheses with evidence
before changing any code.
```

You judge the hypotheses — that is the skill-preserving move.

Lab 3

Your Mission

In Lab 3 you make the pipeline prove itself:

- Write **characterization tests** that pin `clean()`'s current behavior
- Build the statistics module test-first — red, green, refactor
- Run one real debugging hunt with the protocol, not paste-and-pray

Lab 3 — Testing the Pipeline

Continue Your Journey

Next Up

- You can now make an agent prove its work instead of merely claiming it
- Next: keeping that work safe — permissions, secrets, and review
- Lab 3 — Testing the Pipeline starts now
- Lecture VI — Working Safely with AI Code — 10:45
- Course literature and references

Bibliography

[1] J. H. Shen and A. Tamkin, "How AI Impacts Skill Formation." Accessed: July 09, 2026. [Online]. Available: https://arxiv.org/abs/2601.20245