Coding & Development / Code Reviews

Analyze what code paths are missing tests and suggest specific unit or integration tests to add.
Difficulty: Intermediate → Advanced
Model: GPT-4 / Claude / Gemini
Use Case: Test Coverage, Quality Assurance, Testing Strategy
Updated: May 2026
Why This Prompt Exists
Most code has tests — but not for the important edge cases.

You get:

- tests that cover happy paths only (not edge cases)
- error handling code with no tests
- boundary conditions (empty lists, zero values) untested
- no tests for the code that’s most likely to break
- high coverage percentage but low confidence

But test coverage is not about percentage.

It is about risk coverage.

- Normal case: typical inputs, expected outputs
- Edge cases: empty, zero, null, boundary values
- Error cases: invalid inputs, exceptions, timeouts
- Boundary conditions: array bounds, off-by-one
- Concurrency: race conditions, deadlocks

Without coverage assessment, you have false confidence.

This framework forces AI to identify missing tests.

The Prompt

```
Assume the role of a testing strategist who identifies missing test
coverage.

Your task is to recommend tests for uncovered code paths.

Generate:

1. CODE PATHS MISSING TESTS
   - Specific functions/conditions without tests

2. FOR EACH MISSING PATH:
   - Input that would trigger it
   - Expected output/behavior

3. TEST RECOMMENDATION
   - Unit test for isolated logic
   - Integration test for component interaction

4. TEST CODE EXAMPLE
   - Ready-to-use test code

5. PRIORITY (HIGH/MEDIUM/LOW)

INPUTS:
```

```
Function/Code (paste):
[PASTE CODE]

Existing Tests (if any):
[PASTE OR DESCRIBE]

Criticality of This Code:
[HIGH / MEDIUM / LOW]

Known Bug History (if any):
[LIST OR "NONE"]

RULES:
- Check for untested error handling (try/catch blocks)
- Check for untested edge cases (empty lists, null, zero)
- Check for untested boundary conditions (array bounds, off-by-one)
- Check for untested default branches (else/switch default)
- Check for untested rare conditions (timeout, network error)
- Priority: error handling > edge cases > normal cases
- Suggest parameterized tests for multiple similar cases
```

How To Use It

- Test error handling first — it’s where bugs hide.
- Edge cases (empty lists, null, zero) are more important than more happy path tests.
- Use parameterized tests to cover multiple cases with one test function.
- Priority: error handling > edge cases > boundary conditions > normal cases.
- Don’t aim for 100% coverage — aim for high-risk coverage.

Example Input

Function/Code:
def divide(a, b):
if b == 0:
raise ValueError(“Cannot divide by zero”)
return a / b

Existing Tests: Test for normal division (4 / 2 = 2)

Criticality of This Code: HIGH (used in payment calculations)

Known Bug History: None yet

Why It Works
Most tests miss critical edge cases.

This framework improves outcomes by forcing:

- missing path identification (detection)
- input specification (reproducibility)
- test recommendation (action)
- code example (execution)
- priority ranking (focus)

Great coverage assessment doesn’t track percentage — it tracks risk.

Build Better AI Systems

Subscribe for advanced prompt engineering, AI coding tools, code review frameworks, and practical strategies for developers and engineers.

Carefully engineered prompts for people doing real work.

Share this:

- Share on Facebook (Opens in new window) Facebook
- Share on X (Opens in new window) X

See also The Performance Reviewer