Software Engineering

UE23CS341A

5th Semester, Academic Year 2023

Date: 10/11/2025

Name: Roshini Ramesh	SRN: PES1UG23CS488	Section: H

LAB 7: STATIC CODE ANALYSIS

Objective

- Understand the difference between code coverage and branch coverage.
- Use coverage to measure, analyse, and improve test coverage.
- Interpret coverage reports and identify untested paths in code.
- Practice writing test cases that cover edge cases and decision branches.

Learning Outcomes

- Understand Code Coverage Concepts: You will differentiate between line coverage, branch coverage, path coverage, and function coverage to measure test completeness.
- Apply Coverage Analysis Tools: You will use Python's coverage library to generate reports, interpret coverage metrics, and identify untested code paths.
- Design Comprehensive Test Cases: You will write targeted test cases to achieve minimum 90% coverage by addressing edge cases and decision branches.

Introduction

What is Code Coverage?

Code coverage measures how much of your source code is executed during testing.

Types of coverage:

Type	Meaning
Line Coverage	How many lines of code were executed
Branch Coverage	How many branches (e.g., if, else) were followed
Path Coverage	How many unique paths through code were tested
Function Coverage	How many functions were invoked during tests

Why It Matters:

- High coverage improves confidence in correctness.
- Helps catch untested paths and dead code.
- Encourages test completeness, especially in critical systems

Folder Structure:

PES1UG23CSXXX/

|— order_processor.py # Main code to test

|— test_processor_CSXXX.py # Incomplete starter test file, replace XXX with SRN

|— Instructions Manual

RENAME FOLDER and test_processor file TO YOUR SRNS BEFORE RUNNING.

Steps:

Base Iteration:

1. **Install dependencies:**
   pip install coverage

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$ pip install coverage
Collecting coverage
  Downloading coverage-7.11.3-cp313-cp313-win_amd64.whl.metadata (9.3 kB)
Downloading coverage-7.11.3-cp313-cp313-win_amd64.whl (220 kB)
Installing collected packages: coverage
Successfully installed coverage-7.11.3
```

2. **Run the original test file given to you:**
   python -m coverage run -m pytest test_processor_CS488.py

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$ python -m coverage run -m pytest test_processor_CS488.py
============================= test session starts ==============================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\Roshini\Documents\PES\Sem 5\SE\PES1UG23CS488_RoshiniRamesh_Lab7
plugins: anyio-4.11.0, hypothesis-6.145.1
collected 2 items

test_processor_CS488.py ..                                               [100%]

============================== 2 passed in 1.37s ===============================
```

3. **python -m coverage report -m**

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$ python -m coverage report -m
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
order_processor.py           22     15    32%   4, 9-15, 19-29
test_processor_CS488.py       6      0   100%
-------------------------------------------------------
TOTAL                        28     15    46%
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$
```

4. **python -m coverage html**

Coverage report: 46%

Files | Functions | Classes

coverage.py v7.11.3, created at 2025-11-10 12:06 +0530

File ▲	statements	missing	excluded	coverage
order_processor.py	22	15	0	32%
test_processor_CS488.py	6	0	0	100%
Total	**28**	**15**	**0**	**46%**

coverage.py v7.11.3, created at 2025-11-10 12:06 +0530

Coverage report: 46%

Files | Functions | Classes

coverage.py v7.11.3, created at 2025-11-10 12:06 +0530

File ▲	function	statements	missing	excluded	coverage
order_processor.py	calculate_discount	11	6	0	45%
order_processor.py	update_order_status	9	9	0	0%
order_processor.py	*(no function)*	2	0	0	100%
test_processor_CS488.py	test_regular_low_amount	1	0	0	100%
test_processor_CS488.py	test_premium_discount	1	0	0	100%
test_processor_CS488.py	*(no function)*	4	0	0	100%
Total		**28**	**15**	**0**	**46%**

coverage.py v7.11.3, created at 2025-11-10 12:06 +0530

Iteration-1:

Modified Test File:

```
import pytest
from order_processor import calculate_discount, update_order_status

#Example test cases:
def test_regular_low_amount():
    assert calculate_discount("regular", 500) == 0

def test_premium_discount():
    assert calculate_discount("premium", 2000) == 200

# Test cases for calculate_discount function

def test_regular_high_amount():
    """Test regular customer with amount > 1000 gets 5% discount"""
    assert calculate_discount("regular", 2000) == 100

def test_regular_exact_threshold():
    """Test regular customer with amount exactly at 1000"""
    assert calculate_discount("regular", 1000) == 0

def test_premium_various_amounts():
    """Test premium customer gets 10% discount regardless of amount"""
    assert calculate_discount("premium", 500) == 50
    assert calculate_discount("premium", 10000) == 1000

def test_vip_low_amount():
    """Test VIP customer with amount <= 5000 gets 10% discount"""
```

```python
    assert calculate_discount("vip", 3000) == 300
    assert calculate_discount("vip", 5000) == 500

def test_vip_high_amount():
    """Test VIP customer with amount > 5000 gets 20% discount"""
    assert calculate_discount("vip", 6000) == 1200
    assert calculate_discount("vip", 10000) == 2000

def test_unknown_customer_type():
    """Test that unknown customer type raises ValueError"""
    with pytest.raises(ValueError, match="Unknown customer type"):
        calculate_discount("unknown", 1000)
    with pytest.raises(ValueError, match="Unknown customer type"):
        calculate_discount("gold", 500)

# Test cases for update_order_status function

def test_pending_paid_order():
    """Test pending order that is paid moves to processing"""
    order = {"status": "pending", "paid": True, "items_available": True}
    result = update_order_status(order)
    assert result == "processing"
    assert order["status"] == "processing"

def test_pending_unpaid_order():
    """Test pending order that is not paid moves to awaiting_payment"""
    order = {"status": "pending", "paid": False, "items_available": True}
    result = update_order_status(order)
    assert result == "awaiting_payment"
    assert order["status"] == "awaiting_payment"

def test_processing_items_available():
    """Test processing order with items available moves to shipped"""
    order = {"status": "processing", "paid": True, "items_available": True}
    result = update_order_status(order)
    assert result == "shipped"
    assert order["status"] == "shipped"

def test_processing_items_unavailable():
    """Test processing order with items unavailable moves to backorder"""
    order = {"status": "processing", "paid": True, "items_available": False}
    result = update_order_status(order)
    assert result == "backorder"
    assert order["status"] == "backorder"
```

```python
def test_order_status_other_states():
    """Test orders with other statuses remain unchanged"""
    order = {"status": "shipped", "paid": True, "items_available": True}
    result = update_order_status(order)
    assert result == "shipped"
    assert order["status"] == "shipped"
```

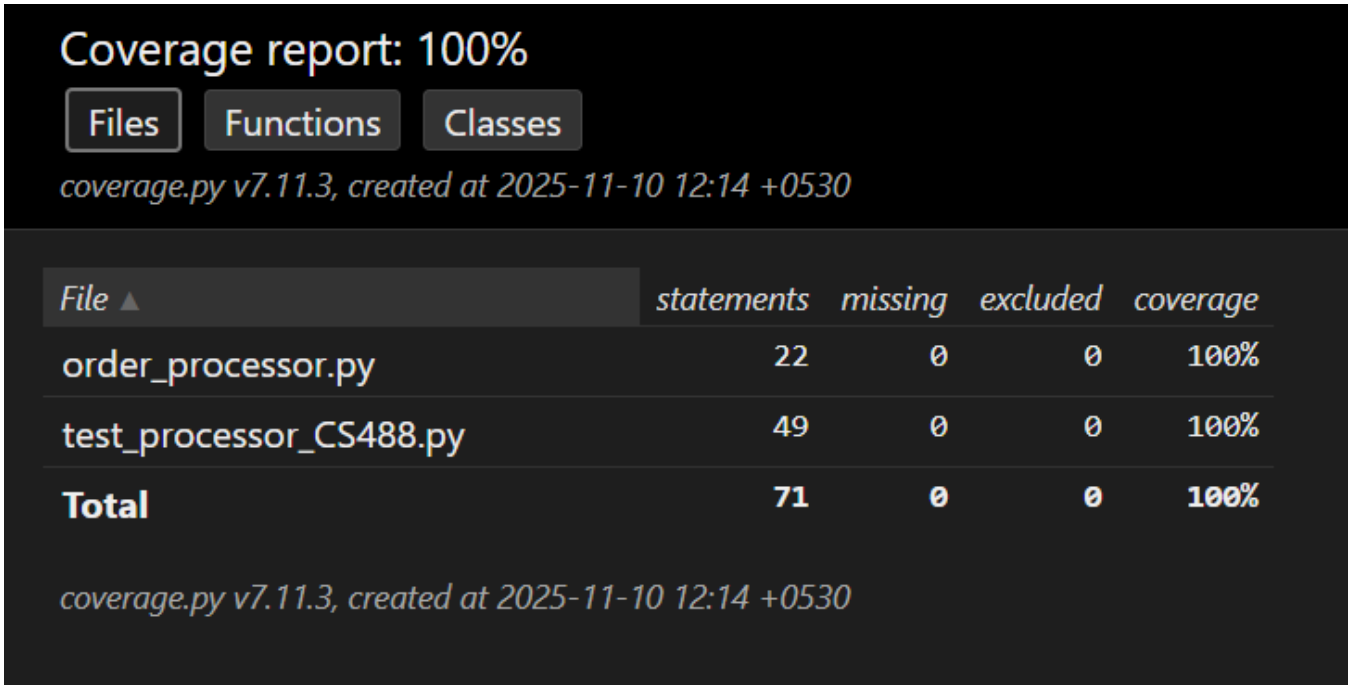
1. **Run the test file after making changes: python -m coverage run -m pytest test_processor_CS488.py**

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$ python -m coverage run -m pytest test_processor_CS488.py
========================================= test session starts =========================================
platform win32 -- Python 3.13.1, pytest-8.4.2, pluggy-1.6.0
rootdir: C:\Users\Roshini\Documents\PES\Sem 5\SE\PES1UG23CS488_RoshiniRamesh_Lab7
plugins: anyio-4.11.0, hypothesis-6.145.1
collected 13 items

test_processor_CS488.py .............                                                        [100%]

========================================== 13 passed in 0.93s ==========================================
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$
```

2. **python -m coverage report -m**

```
rosh@LAPTOP-10GABJ15:~/Documents/PES/Sem 5/SE/PES1UG23CS488_RoshiniRamesh_Lab7$ python -m coverage report -m
Name                      Stmts   Miss  Cover   Missing
-------------------------------------------------------
order_processor.py           22      0   100%
test_processor_CS488.py      49      0   100%
-------------------------------------------------------
TOTAL                        71      0   100%
```

3. **python -m coverage html**



Coverage report: 100%

Files | Functions | Classes

coverage.py v7.11.3, created at 2025-11-10 12:14 +0530

File ▲	function	statements	missing	excluded	coverage
order_processor.py	calculate_discount	11	0	0	100%
order_processor.py	update_order_status	9	0	0	100%
order_processor.py	*(no function)*	2	0	0	100%
test_processor_CS488.py	test_regular_low_amount	1	0	0	100%
test_processor_CS488.py	test_premium_discount	1	0	0	100%
test_processor_CS488.py	test_regular_high_amount	1	0	0	100%
test_processor_CS488.py	test_regular_exact_threshold	1	0	0	100%
test_processor_CS488.py	test_premium_various_amounts	2	0	0	100%
test_processor_CS488.py	test_vip_low_amount	2	0	0	100%
test_processor_CS488.py	test_vip_high_amount	2	0	0	100%
test_processor_CS488.py	test_unknown_customer_type	4	0	0	100%
test_processor_CS488.py	test_pending_paid_order	4	0	0	100%
test_processor_CS488.py	test_pending_unpaid_order	4	0	0	100%
test_processor_CS488.py	test_processing_items_available	4	0	0	100%
test_processor_CS488.py	test_processing_items_unavailable	4	0	0	100%
test_processor_CS488.py	test_order_status_other_states	4	0	0	100%
test_processor_CS488.py	*(no function)*	15	0	0	100%
Total		**71**	**0**	**0**	**100%**

coverage.py v7.11.3, created at 2025-11-10 12:14 +0530

Coverage report: 100%

Files | Functions | Classes

coverage.py v7.11.3, created at 2025-11-10 12:14 +0530

File ▲	class	statements	missing	excluded	coverage
order_processor.py	*(no class)*	22	0	0	100%
test_processor_CS488.py	*(no class)*	49	0	0	100%
Total		**71**	**0**	**0**	**100%**

coverage.py v7.11.3, created at 2025-11-10 12:14 +0530

Reflection:

1. **If the logic in order_processor.py changes, what's your strategy to ensure tests and coverage stay updated?**

   The strategy I would follow would be as follows:

   - Before I start making changes to order_processor.py, I would start by running the existing tests to make sure everything passes. I would also check the current coverage report to see what is already covered.
   - Based on my analysis of the above, I would write tests for all the areas that I feel haven't been covered or are lacking. Once I write the test cases, then I would modify the code as this gives me an outline of exactly what I need to achieve through the modifications and the modifications I introduce would now be testable.
   - After making the changes, I would run pytest to see if any tests fail. Then, I would check the coverage percentage and coverage report to make sure that every aspect of the code has been tested. If the coverage drops, I would go back to identify the uncovered lines and add tests for them. I would make sure that the coverage stays above 90%.
   - Additionally, I would also set up CI/CD pipeline that automatically runs tests on every commit. Using GitHub Actions, I would block merging if tests fail or coverage drops which would protect faulty code from getting pushed

2. **What are the trade-offs between writing more tests for coverage vs writing fewer high-quality tests?**

   There are multiple pros and cons to writing more tests for coverage vs writing fewer high-quality tests.

   If we look at writing more tests for coverage:

   - The pros are:
     - We would get higher coverage percentage (90-100%)
     - We would be able to catch more edge cases
     - We would be able to find bugs we didn't think of
   - But the cons are:
     - It would take us a lot more time to write and maintain
     - Running the tests would be very time-consuming
     - If we change one function, we would have to update 20+ tests

   On the other hand, if we wrote fewer high-quality tests:

   - The pros are:
     - It would be faster to write and run
     - It would be easier to maintain
     - We would be able to focus on important functionality
     - There would be clarity on what each test does

- But the cons are:
    - We might miss edge cases
    - There would be a lower coverage number
    - There might be some bugs might slip through

I believe that there should be a balance in the number of test cases. We must write enough tests to cover all the important branches and edge cases but it is unnecessary to test every single value.

I would target for 80-95% coverage and focus on testing the logic that is most likely to break or is most important to the application or program that we are developing.