The AI-Native Refactorer, Topic 2: Duplicated Code	Student Assignment: Find and Consolidate a Conceptual Duplicate

Overview

In the hands-on, the instructor walked you through finding three flavors of duplication in a Python file. Now you'll do it yourself on a different file — and you'll have to find a conceptual duplicate without anyone telling you it exists.

Estimated time: 10 minutes.

WHAT YOU'LL PRACTICE

Driving the AI through a duplication scan, recognizing when it returns false positives, and consolidating without breaking the existing call sites. This assignment focuses on conceptual duplicates specifically because they are what humans miss most.

The Code You'll Work With

Below is a file with five functions. Two of them are conceptual duplicates with different implementations. Two are structural duplicates. One is genuinely unique. Your AI assistant should be able to identify which is which. Copy this into a file called assignment2.py before you start.

```
from datetime import datetime, timedelta


def total_unread_messages(messages):
    """Returns the count of unread messages."""
    count = 0
    for m in messages:
        if not m.read:
            count += 1
    return count


def count_pending_orders(orders):
    """Returns the count of pending orders."""
    pending = 0
    for o in orders:
        if o.status == "pending":
            pending += 1
```

```
    return pending


def average_response_time(tickets):
    """Returns the average response time in minutes."""
    if not tickets:
        return 0
    total = 0
    for t in tickets:
        total += t.response_minutes
    return total / len(tickets)


def get_overdue_invoices(invoices):
    """Returns invoices that are past their due date."""
    now = datetime.utcnow()
    overdue = []
    for inv in invoices:
        if inv.due_date < now:
            overdue.append(inv)
    return overdue


def find_late_payments(payments):
    """Returns payments where today is after the expected date.

    Note: completely different implementation from get_overdue_invoices,
    but the intent is identical."""
    today = datetime.utcnow().date()
    late = []
    for p in payments:
        days_late = (today - p.expected_date.date()).days
        if days_late > 0:
            late.append(p)
    return late
```

Your Task

Apply the five-step workflow:

Step 1 — SCAN

Ask your AI assistant to classify the five functions into three categories: structural duplicates, conceptual duplicates, and uniques. Write the classification in your README.md with one sentence of reasoning per cluster.

Step 2 — PROTECT

Generate pytest characterization tests for all five functions. Save as assignment2_tests.py and run them — they should all pass against the original code.

Step 3 — CONSOLIDATE

Drive the AI to consolidate each cluster. The structural duplicates (count-where-condition) should become a single generic function. The conceptual duplicates (is-past-date) should pick ONE implementation as canonical. The unique function stays unique. Use the strangler-fig pattern — keep the original names as thin wrappers.

Step 4 — VERIFY

All tests still pass. Confirm by asking the AI to compare your consolidated version to the original and confirm behavior is preserved (with any deliberate bug-fixes called out).

Deliverables

- assignment2.py — your consolidated version with thin wrappers.
- assignment2_tests.py — characterization tests, all passing.
- README.md — under 200 words: classification of the five functions, the canonical helpers you created, and any drift you consciously decided to fix vs preserve.

Self-Assessment Rubric

How to know you're done

✓ You correctly identified the conceptual duplicate (get_overdue_invoices and find_late_payments).

✓ You correctly identified the structural duplicate (the two count-with-condition functions).

✓ You correctly identified the unique function (average_response_time).

✓ All characterization tests pass against your consolidated code.

✓ Your canonical functions are named clearly enough that a new contributor would use them by default.

Common mistakes to watch for

✗ Trying to consolidate the unique function with the others — "close enough" is not duplication.

✗ Forgetting that average_response_time uses len() — the count helpers don't.

✗ Asking the AI to consolidate without first generating tests.

✗ Removing the wrapper functions before any callers have been migrated.