The AI-Native Refactorer, Topic 4: Tight Coupling	Student Assignment: Make a Tightly-Coupled Class Testable

Overview

In the hands-on, the instructor walked through inverting dependencies on an OrderProcessor. Now you'll do the same on a different class — a NewsletterScheduler — that has two concrete dependencies and one hidden one. The goal is the same: a unit test that runs in milliseconds with fakes.

Estimated time: 10 minutes.

WHAT YOU'LL PRACTICE

Identifying both explicit and hidden dependencies; introducing Protocols; injecting dependencies via the constructor; and writing the first unit test that proves the refactor worked.

The Code You'll Work With

Below is a NewsletterScheduler that has two concrete dependencies in its constructor AND a hidden one — it reads the current time via datetime.utcnow() inside its main method, which makes it impossible to test deterministically. Copy it into assignment4.py.

```
import os
from datetime import datetime


class MailgunClient:
    def __init__(self):
        self.api_key = os.environ.get("MAILGUN_KEY", "key-fake")
        self._sent = []

    def send(self, to, subject, body):
        self._sent.append({"to": to, "subject": subject, "body": body})
        return True


class SubscriberDb:
    def __init__(self):
        self.url = os.environ.get("SUBSCRIBER_DB", "sqlite://subscribers.db")
        self._subscribers = []  # in-memory stub
```

```
    def find_active(self):
        return [s for s in self._subscribers if s.get("active")]

    def add(self, subscriber):
        self._subscribers.append(subscriber)


class NewsletterScheduler:
    """Sends a weekly newsletter to active subscribers — but only on Mondays."""

    def __init__(self):
        # Tight coupling: two concrete dependencies created here.
        self.email = MailgunClient()
        self.db = SubscriberDb()

    def send_weekly_newsletter(self, subject, body):
        # Hidden time-coupling: reads the current time directly.
        today = datetime.utcnow()
        if today.weekday() != 0:  # 0 = Monday
            return {"status": "skipped", "reason": "not Monday"}

        subscribers = self.db.find_active()
        sent_count = 0
        for sub in subscribers:
            self.email.send(sub["email"], subject, body)
            sent_count += 1

        return {"status": "sent", "count": sent_count}
```

Your Task

Apply the five-step workflow.

Step 1 — SCAN

Ask the AI to list every dependency of NewsletterScheduler, including the hidden ones. There are three: MailgunClient, SubscriberDb, and the system clock (datetime.utcnow). The clock dependency is the one most students miss.

Step 2 — PROTECT

Generate characterization tests against the existing code. Note: you'll find this awkward — the test for the not-Monday branch only works on six days out of seven. That awkwardness IS the proof that time-coupling is a problem.

Step 3 — INVERT

Introduce three things:

- A Protocol for each dependency — EmailClient, SubscriberRepository, and Clock (with a now() method).
- Updated constructor that accepts three parameters, all typed against the Protocols.
- A wire() function at the bottom that produces a production NewsletterScheduler with a SystemClock implementation.

Step 4 — VERIFY

Write three unit tests with fakes (FakeEmail, FakeDb, FakeClock). The third test should pass a FakeClock returning a Monday to prove the newsletter sends, and a FakeClock returning a Tuesday to prove it skips. The tests should run in milliseconds and not depend on what day it actually is.

Deliverables

- assignment4.py — your refactored NewsletterScheduler with three Protocols, DI, and a wire() function.
- assignment4_tests.py — three unit tests with fakes, all passing, all deterministic.
- README.md — under 200 words: list the three dependencies, the three Protocols you defined, and explain in one sentence why FakeClock matters more than FakeEmail or FakeDb here.

Self-Assessment Rubric

How to know you're done

✓ You identified the clock as a hidden dependency.

✓ Three Protocols defined: EmailClient, SubscriberRepository, Clock.

✓ NewsletterScheduler's constructor accepts all three.

✓ Production wiring lives in a wire() function, not in NewsletterScheduler.

✓ Tests pass on ANY day of the week — Monday or otherwise.

Common mistakes to watch for

✗ Missing the clock dependency. If your tests can fail on a Sunday, you missed it.

✗ Putting wire() inside NewsletterScheduler — it belongs OUTSIDE the class.

✗ Making the fakes inherit from the concrete classes. They should only need to satisfy the Protocol.

✗ Reading os.environ inside the class. All environment reads belong in wire().

Course Wrap-Up

This is the final assignment of The AI-Native Refactorer. You've now practiced the full five-step refactoring workflow across all four major refactoring targets. The combination of characterization tests, AI-driven scanning, scoped prompts, and named refactoring techniques is the toolkit that turns refactoring from a risky chore into a 15-minute habit. If you want to keep going, the full paid course at https://TheAI-NativeEngineer.com covers another twelve patterns and AI-driven code review.