Create Simulator Tests

Apply the Hexagonal Architecture

Problem and Solution Overview

This recipe just states how to execute a solution. Read **DevOps #9 — Ease Integration with Hexagonal Architecture** to understand the specific problem we are solving and the solution approach.

Access **DevOps Series** for getting your software clean for the pipeline.

This recipe helps you verify that your code uses its dependencies correctly, without running any integration tests.

1. Create a Simulator Stub that Ignores Contract Tests
2. Use Simulator Stub to Write a Failing Boundary Code Test
3. Write Enough Simulator to Pass Both Tests

Create a Simulator Stub that Ignores Contract Tests

The goal is to implement just enough Simulator to test the boundary code. We implement the Port parts used by this code, nothing more. That implementation needs to be fully valid — matching other Adapters.

First create an empty Simulator and make sure it doesn't break the tests, even though it can't pass them yet.

Steps:

1. Create a new, empty Adapter for your Port.
    A. Optimally your Adapters bind to the Port by composition.
        Create a new Adapter that returns a Port instance with everything bound to a `noop`.
    B. Use interface inheritance if your design limits you.
        Create an Adapter that implements your Port with an empty body for every method.
    C. Return default values for any required returns.
    D. Name your Adapter based on its real-world meaning, as if it were product code.
        Avoid the word `Simulator` in the name if you can.

Example names might be `FileSystem_InMemory`, `ObjectStore_Cache`, or `StoppedClock`.

2. Put your Adapter under Contract Test.
    A. Create a new test implementation that inherits from your `PortTests`.
    B. Implement `CreateTestSubject` to implement your Adapter and return its bound Port.
    C. Run the tests and verify that they all fail.
    D. Override every test and ignore it. For example:

    E. `[Ignore] public override void WritingAFile_Should_AllowReadingIt() { base.WritingAFile_Should_AllowReadingIt(); }`
    F. Run the tests and verify they are all skipped.
    G. Commit.

Use Simulator Stub to Write a Failing Boundary Code Test

Now use your Simulator to write one test for your boundary code. This step is the same whether your boundary code already exists or you are TDDing it into existence.

Steps:

1. Create one test method and give it a good name.
2. Instantiate your Simulator as a local.
3. Call your boundary code and pass it the Simulator's Port directly.
    Don't use Dependency Injection, just pass in an argument or constructor argument.
4. Make one assertion about the result.
    This probably asserts that the Simulator is in some state, such as having a file written in a specific place with specific contents.
5. Mark the test as ignored.
6. Commit.

Write Enough Simulator to Pass Both Tests

This step uses the failing tests to guide our Simulator implementation. If you are also TDDing your product code, then the Simulator test will guide that implementation as well.

Steps:

1. Un-ignore the boundary code test.

2. Implement some boundary code.
    A. Use the simplest implementation that will make progress towards passing this one test.
    B. You will need to work in very tiny steps, as each step also requires implementing the Simulator and we still want to check in frequently.
    C. Sometimes it is simplest to start with the assertion, sometimes with the action. Don't try to get both the action and assertion to work in the same step.
3. Use Find Usages on every Port method you call to find all Contract Tests you will have to pass to make this work.
    A. If that is more than 3 tests, then revert and try again with even less boundary code in step 2.
    B. Un-ignore these contract tests.
    C. Use them to TDD just enough Simulator to pass these tests.
    D. Ignore the boundary test again. Make sure all tests pass or are skipped.
    E. Commit just the contract test and the Simulator.
4. Check for progress on your boundary code and its test.
    A. Un-ignore the boundary test.
    B. Check if there is some way that you could make it partially pass, such as by running the first couple of lines and then throwing an `IgnoreException` (if your test framework supports this).
    C. Commit if there is any partially-run test or tested code that you could commit.
5. Repeat from step 2 until the test fully passes.

Now you can go back to Write a Failing Boundary Code Test and continue to TDD your Simulator and boundary code together until you are done.