Information Sheet 3

Smells: TDD Anti-patterns, Causes and Corrections

TDD & BDD – Design Through Testing

TDD Anti-Patterns

A "Smell" in TDD is something ugly about the test that tells you something is wrong.

How can you spot them?

What problems do they cause?

What corrective actions can you take?

Heed these Smells. Clarity in your tests is paramount

The Liar

Passes all tests, but doesn't actually assert anything useful

Causes:

Commonly seen in teams that chase "coverage".

Commonly seen in teams that don't practice "Red-Green-Refactor".

Problems:

Illusion of coverage.

Waste! Effort for no value.

Corrections:

Delete!

Practice Test FIRST – Always see the test fail, before making it pass!

Excessive Setup

Requires lots of work to get the code ready to test.

Causes:

Indicates poor separation of concerns in the code under test.

Problems:

Highly-coupled and so very fragile.

Over time projects grind to halt.

Inflexible.

Difficult to understand and maintain.

Corrections:

Improve abstraction in SUT.

Improve Separation of Concerns: "One Class, One Thing. One Method, One Thing!"

Practice Test FIRST: Writing the Test first encourages you to simplify setup.

The Giant

Comprised of many lines of code and makes multiple assertions

Causes:

Commonly caused by "Test-After" Unit-testing.

Often implemented as "Component-Tests".

Problems:

Intent is very hard to determine, does not really document behaviour.

Highly-coupled and so very fragile, over time projects grind to halt.

Corrections:

Decompose into multiple, simpler, Test-Cases.

Assert one outcome per-per-test.

The Mockery

Uses so many mocks and stubs that the SUT is not tested at all

Causes:

Over-complex setup (See "Excessive Setup")

Poor Coupling in SUT.

Problems:

Not readable! Poor as documentation of intent.

Doesn't actually test anything - pure wasted effort!

Difficult to understand and maintain.

Corrections:

Review Design of SUT.

Improve Abstraction and Separation of Concerns.

Practice Test-First Development.

The OS Evangelist

Relies on a specific operating system environment in which to run.

e.g. Runs on Windows and assumes the Windows newline sequence, then fails when run in Linux

Causes:

Lazy Test Design.

Problems:

Limits utility of the test.

This is a form of coupling, it may not matter, but it certainly limits flexibility.

Corrections:

Make tests simple and focused.

Assert only what is needed.

Limit coupling even if you don't know that you will need it - loose-coupled code, and tests, are better!

The Local Hero

Depends on something outside the scope of the test so only runs in one place (e.g. local dev machine).

Causes:

Test is not atomic.
Poor Isolation.
Poor Abstraction.

Problems:

Environment Specific.
Poor Isolation often prevents parallelism in tests (contends on shared resources).

Corrections:

Make tests atomic.
Improve Abstraction in SUT.
Improve Separation of Concerns in SUT.

The Slow Poke

A Test Case that runs extremely slowly

Causes:

Accessing complex resources (e.g. disk, DB, Network).
Symptom of poor design in SUT.
Often a result of Poor Separation of Concerns.
Symptom of a team that don't understand TDD.
Over-testing cases, lengthy iteration is almost never necessary in unit tests.

Problems:

Tests are expensive and often flaky.
Tests don't usefully Document the system.
Impossible to determine intent of test without understanding all the test code and the code of the SUT.

Corrections:

Improve the Separation of Concerns in the SUT.
Isolate and abstract access to complex infrastructure – test to the abstraction not the real thing.
Make test cases atomic.
Avoid iteration in test cases. Use "Chicken Counting" (0, 1 or Many!).

The Inspector

Violates encapsulation to make assertions

Causes:

Striving for 100% test coverage.

Anaemic Objects.

No, or improper, use of dependency injection.

Not practising "Tell Don't Ask".

Problems:

Tight-coupling between Test and SUT.

Highly-coupled and so very fragile, over time projects grind to halt.

Corrections:

Never compromise encapsulation for testing.

Improve abstraction in SUT.

Improve Separation of Concerns in SUT.

The Nitpicker

Asserts on a complete output when it only cares about a small part of it.

Causes:

Lazy Test Design.

Problems:

Tight-coupling between Test and SUT.

Very fragile in the face of change - leads to increase in maintenance burden and over time can halt progress in a project.

Corrections:

Focus each Test on a specific assertion.

Think about the goal of every test and express it in the test.

The Sequencer

Depends on items in an unordered list appearing in the same order during assertions

Causes:

Poor Test Design.

Naive view of the code.

Problems:

Intermittent Test.

Can hide real errors.

Corrections:

Consider use of appropriate data structures in SUT and Test

Understand the properties of the data-structures in-use

The Loudmouth

Clutters up the console with diagnostic messages even when tests are passing

Causes:

Lazy Test Design.

Poor abstraction in SUT.

Sometimes logging was added during test-design and then left behind.

Problems:

Noise!

Hides real errors.

Adds unnecessary effort to debugging errors.

Corrections:

Keep logs, and console, clean so that information is presented only when needed.

Make assertions of failure clear and unambiguous.

The Free Ride

Assertion added to an existing test case rather than writing a new case

Causes:

Lazy Test Design.

A false sense of optimisation for the test code.

Problems:

Makes it harder to interpret failures.

Documentation is poorer.

Often means that the test name is now inappropriate.

Corrections:

Make test cases atomic.

Have one assertion per test-case.

Create the new test case!

The Dodger

Makes lots of assertions of minor (and easy to test) side-effects, but never tests the core behaviour

Causes:

Lazy Test Design.

Tests Written After the SUT.

Developers not understanding the problem.

Problems:

Illusion of Test Coverage.

Will never catch important real bugs.

Often means Test is tightly-coupled to SUT.

Correction:

Assert the expected failure.

The Stranger

Doesn't belong in the suite it is part of.

(It is really testing an entirely different piece of code, or unrelated behaviour)

Causes:

Lazy Test Design.

Poor Encapsulation and Separation of Concerns.

Feature Envy.

Probably indicates SUT is highly-coupled.

Can result from refactoring, introducing a new class to SUT and not refactoring tests to pull new test-suites out.

Problems:

Poor documentation of intent.

Hard to navigate tests.

May make code hard to refactor.

Corrections:

Make tests simple and clearly focused on outcomes.

Use "Separation of Concerns" to guide design decisions.

When refactoring to introduce new concepts, consider tests cases and the suites they should belong to after the change.

The Enumerator

Tests are named numerically, e.g. "Test1", "Test2", "Test3"

Causes:

Lazy Test Design.

Problems:

Intent is not known without reading, and understanding the code.

Commonly goes hand-in-hand with "Excessive Setup", "The Giant" and "The Dodger".

This is a symptom of teams that don't buy, or don't understand, testing in general and TDD specifically.

Corrections:

Name tests for their intent.

Adopt a naming convention and stick to it, e.g: "shouldDoSomethingThatICareAbout"

The Secret Catcher

A Test Case that makes no assertions but relies on an exception to be thrown to cause it to fail

Causes:

Lazy Test Design

Problems:

If the SUT changes to no-longer throw the exception, the test still passes.

Corrections:

Assert the expected failure

Think about the goal of every test and express it in the test

The Hidden Dependency

Depends on data to have been populated or initialised before the test runs

Causes:

Test is not atomic.

Poor Isolation.

Poor Abstraction.

Problems:

Intermittent Test.

Prevents running tests in parallel, or if dependency is on another test, in a different order.

Corrections:

Make Tests Atomic

Provide an abstraction for accessing a dependency; use mocking and dependency-injection.

Success Against All Odds

Written to pass first rather than fail first.
As a result it always passes even when it shouldn't

Causes:

Lazy Test Design.

Not practicing Test-First and Red-Green-Refactor.

Problems:

Test is useless.

Provides the illusion of coverage.

Corrections:

Always "test the test" and see it fail before you make it pass.

Practice true TDD - Test-First.

If you do need to retro-fit a test after the code exists, break the code to see the test fail.

The One

A combination of several other anti-patterns –
One Test Case that tests the entire set of functionality of an object, module or function

Causes:

Often the result of Unit testing after the fact, rather than TDD.

Symptom of teams that don't believe, or don't understand testing in general and TDD specifically.

Problems:

Makes it harder to interpret failures.

Documentation is poorer.

Impossible to determine intent of test without understanding all the test code and the code of the SUT.

Corrections:

Make test cases atomic.

Have one assertion per test-case.

Further Reading

TDD Antipatterns from Codurance: https://bit.ly/35Nhz2a