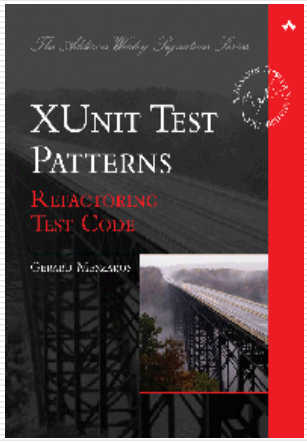
xUnit Test Patterns



Adapted by JP from:

Negar Koochakzadeh

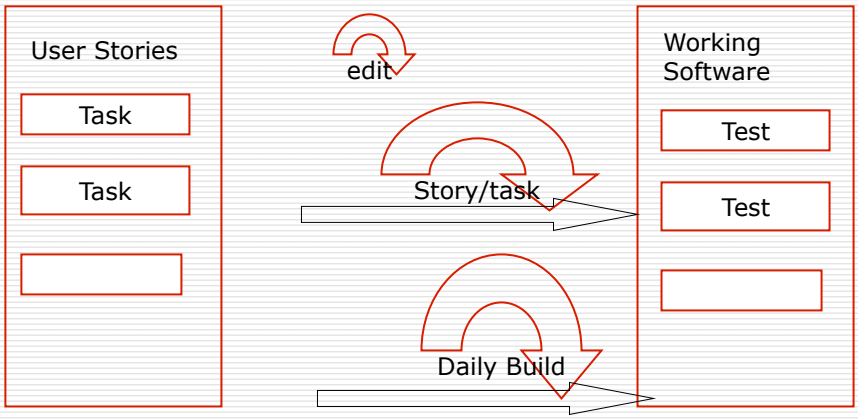
Venkat Mantripragada

AND

Meszaros's book

1

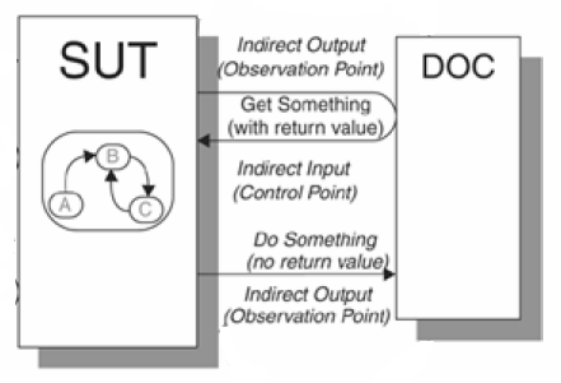
Agile Development Cycles



2

System Under Test

- It may have Depended-on components (DOCs)



3

Four phase testing

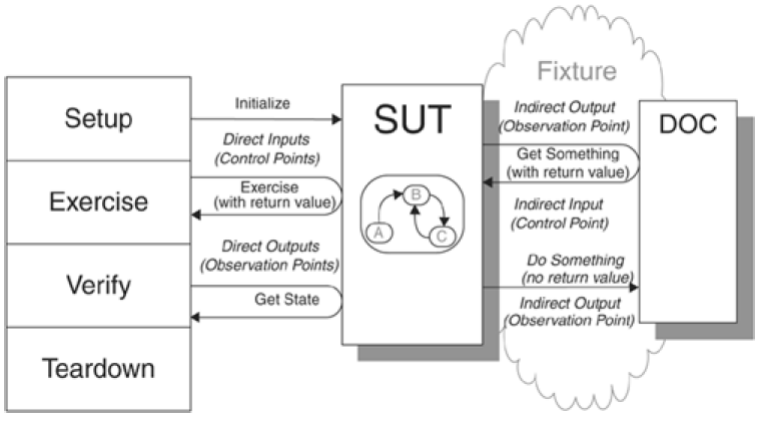


Image From: xunit Test Patterns, G. Meszaros

4

Test Case 'Generation'

- Recorded Test (esp. for GUI)
  - When to use:
    - We do not expect a lot of changes for the system.
- Scripted Test
  - We write it by hand
- 'Automated' Test case generation
  - Model-based testing

In this presentation we discuss the art of writing tests.

5

Recorded Test

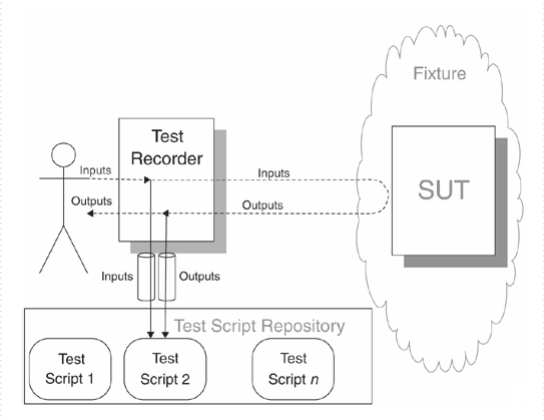


Image From: xunit Test Patterns, G. Meszaros

6

Scripted Test

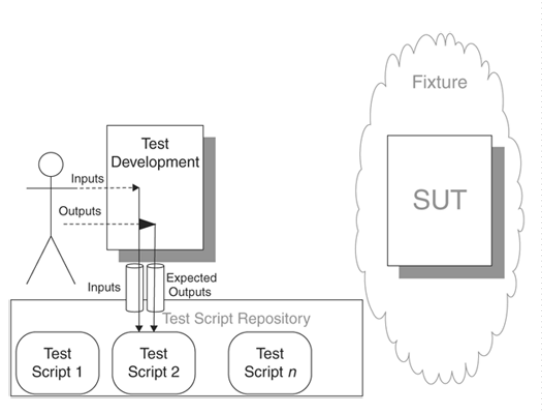


Image From: xunit Test Patterns, G. Meszaros

7

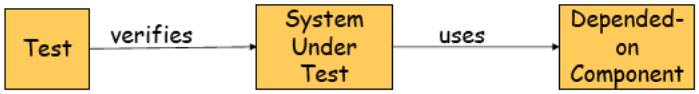
Right Attitude towards Testing?

8

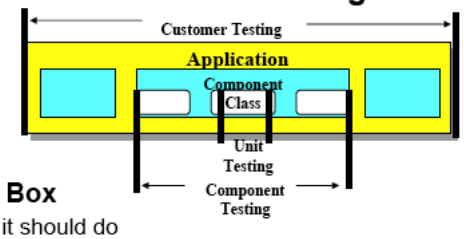
Image SRC: www.dilbert.com

Terminology

- **Test vs SUT vs DOC:**



- **Unit vs Component vs Customer Testing**



- **Black Box vs White Box**
  - Black box: know what it should do
  - White box: know how it is built inside

Is the BB vs. WB distinction relevant to TDD??

9

Coding Objectives Comparison

	Production	*Testware*
Correctness	Important	Crucial
Maintainability	Important	Crucial
Execution Speed	Crucial	Somewhat
Reusability	Important	Somewhat
Flexibility	Important	Not
Simplicity	Important?	Crucial
Ease of writing	Important?	Crucial

From: xunit Test Patterns, G. Meszaros

10

A Sobering Thought

Expect to have just as much test code as production code!

The Challenge: How To Prevent Doubling Cost of Software Maintenance?

Answer: Test Automation!!

11

Goals of test automation

- Improve quality
- Understand the SUT
- Reduce the risk (of error, of omission, of delay)
- Easy to run
- Easy to write
- Easy to maintain

12

Why are Tests so Important?

- **Tests need to be maintained along with rest of the software.**
- **Testware must be much easier to maintain than the software, otherwise:**
  - It will slow you down
  - It will get left behind
  - Value drops to zero
  - You'll go back to manual testing

Critical Success Factor:

Writing tests in a maintainable style

13

Economics of Maintainability

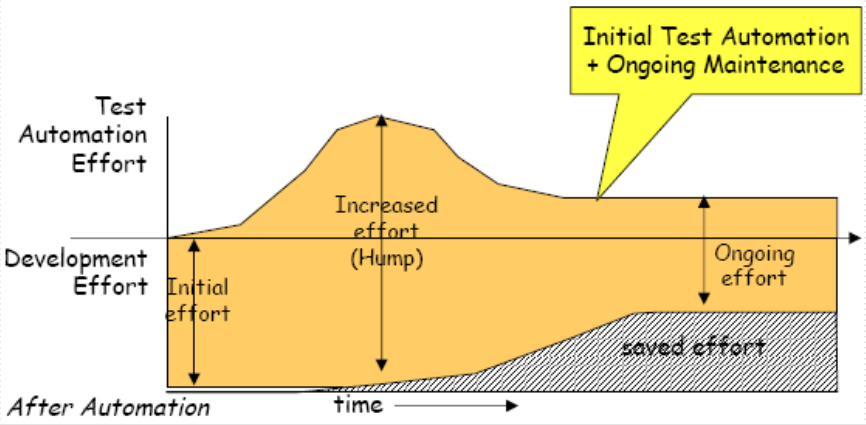


Image From: xunit Test Patterns, G. Meszaros

14

Principles in test automation

- TDD: Write the test first
- Each test should be:
  - Small and simple
  - Independent of other tests
  - Repeatable
  - Self-checking → Fully Automated
- First do "State verification" (controllability)
  - Make sure you are in the right state to test what you want to test

  and then "Behavior Verification" (observability)

15

Benefits of Automated Tests

- **Before code is written**
  - Tests as Specification
- **After code is written**
  - Tests as Documentation
  - Tests as Safety Net (Bug Repellent)
  - Defect Localization (Minimize Debugging)
- **Minimize Cost of Running Tests**
  - Fully Automated Tests
  - Repeatable Tests
  - Robust Tests

16

? Which part can be automated?

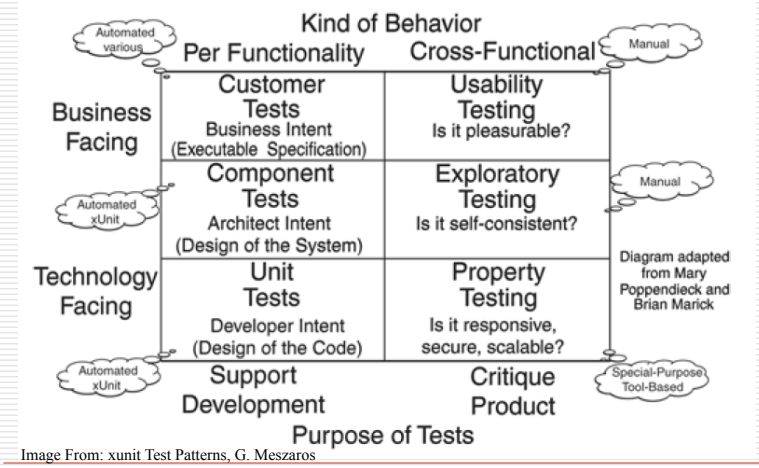


Image From: xunit Test Patterns, G. Meszaros

17

Common features of XUnit family

- Specify a test as a test Method
- Specify the expected results within the test method in the form of calls to Assertion Methods
- Aggregate the tests into test suites that can be run as a single operation
- Run one or more tests to get a report on the results of the test run.

18

Some XUnit Tools

- C: CUnit, Check, RCUNIT
- C++: CPPUnit, CppUnitLite, CxxTest
- Delphi: DUnit
- Java: JUnit, TestNG
- JavaScript: JSUnit
- Matlab: mlUnit
- .Net: csUnit, NUnit, MbUnit, xUnit
- PHP: PHPUnit, Testilence
- Python: PyUnit, Trial

19

A Recipe for Success

1. Write some tests
   - start with the easy ones!
2. Note the Test Smells that show up
3. Refactor to remove obvious Test Smells
   - Apply appropriate xUnit Test Patterns
4. Write some more tests
   - possibly more complex
5. Repeat from Step 2 until:
   - All necessary tests are written
   - No smells remain

20

What is a Test Smell?

- A Smell is a symptom of a problem in a test code.
- Not necessarily the actual cause
  - ✓There may be many possible causes for the symptom
  - ✓Some root causes may contribute to several different smells
- Not all problems are considered as smells
  - ✓Smells must pass "Sniffability test" (ie be obvious to detect)

21

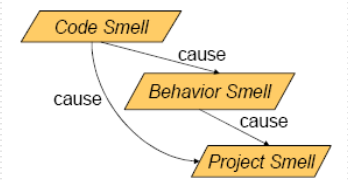
What is a Xunit Test Pattern?

- **A "test pattern" is a recurring solution to a test automation problem**
  - E.g. A "Mock Object" solves the problem of verifying the behavior of an object that should delegate behavior to other objects
- **Test Patterns occur at many levels:**
  - Test Automation Strategy Patterns
    - » **Recorded Test vs Scripted Test**
  - Test Design Patterns
    - » **Implicit SetUp vs Delegated SetUp**
  - Test Coding Patterns
    - » **Assertion Method, Creation Method**
  - Language-specific Test Coding Idioms
    - » **Expected Exception Test, Constructor Test**

22

Kinds of Test Smells

- Code Smells
  - Recognized by looking at test code
- Behavior smells
  - Effects the outcome of tests as they execute
- Project Smells
  - Recognized by project managers. Root cause can be one or more code/ behavioral smells



23

Code Smells

A problem visible when looking at test code:

- Tests are hard to understand
- Tests contain coding errors that may result in
  - Missed bugs
  - Erratic Tests
- Tests are difficult or impossible to write
  - No test API on SUT
  - Cannot control initial state of SUT
  - Cannot observe final state of SUT
- Sniff Test
  - Problem must be visible to test automater or test reader

24

General Code Smells

- Obscure Test
- Conditional Test Logic
- Hard-to-Test Code
- Test Code Duplication
- Test Logic in Production

Obscure Test

"Test is hard to understand"

- Common Causes:

Verbose Test
- So much test code that it obscures the test intent

Eager Test
- Several tests merged into one Test Method

General Fixture
- Fixture contains objects irrelevant for this test

Obtuse Assertion
- Using the wrong kind of assertion

Hard-Coded Test Data
- Lots of "Magic Numbers" or Strings used when creating objects

Eager Test

```
public void testFlightMileage_asKm2() throws Exception
{ // set up fixture // exercise constructor
    Flight newFlight = new Flight(validFlightNumber);
// verify constructed object
    assertEquals(validFlightNumber, newFlight.number);
    assertEquals("", newFlight.airlineCode); assertNull(newFlight.airline);
    newFlight.setMileage(1122); // set up mileage
    int  actualKilometres = newFlight.getMileageAsKm(); // try mileage translator
    int expectedKilometres = 1810;
    assertEquals( expectedKilometres, actualKilometres);
    newFlight.cancel(); // now try it with a canceled flight
    try { newFlight.getMileageAsKm(); fail("Expected exception");  }
          catch (InvalidRequestException e)
    { assertEquals( "Cannot get cancelled flight mileage", e.getMessage()); }

}
```

Testing too many functionalities

Irrelevant Information

```
public void testAddItemQuantity_severalQuantity () {
final int QUANTITY = 5;
Address billingAddress = new Address("1222 1st St SW",
"Calgary", "Alberta", "T2N 2V2", "Canada");
Address shippingAddress = new Address("1333 1st St SW",
"Calgary", "Alberta", "T2N 2V2", "Canada");
Customer customer = new Customer(99, "John", "Doe", new
        BigDecimal("30"), billingAddress, shippingAddress);
Product product = new Product(88, "SomeWidget", new
BigDecimal("19.99"));
Invoice invoice = new Invoice(customer);
// Exercise SUT
invoice.addItemQuantity(product, QUANTITY);
```

Hard to determine Which val Effects outcome

Obscure Test

Indirect Testing

- Interacting with the SUT via other software
- A cause of Fragile Tests (Behavior Smell)

Mystery Guest

- Lots of "Magic Numbers" or Strings used as keys to database.
- "Lopsided" feel to tests
  - either Setup or Verification of outcome is external to test

29

Conditional Test Logic

- Tests containing conditional logic (IF statements or loops)
- Hard to verify correctness.
- A cause of Buggy Tests (Project Smell)

30

Conditional Test Logic

```
actual = null; // verify Vancouver is in the list
    i = flightsFromCalgary.iterator();
    while (i.hasNext())
  {
     FlightDto flightDto = (FlightDto)  i.next();
    if   (flightDto.getFlightNumber().equals
    ( expectedCalgaryToVan.getFlightNumber())) {
          actual = flightDto;
          assertEquals("Flight from Calgary to Vancouver",
              expectedCalgaryToVan, flightDto); break; }
     }
}
```

31

Test Code Duplication

- Same code sequences appear many times in many tests
- More code to modify when something changes
- A cause of Fragile Tests (Behavior Smell)

32

Test Code Duplication

```
public void testInvoice_addTwoLineItems_sameProduct()
{   Invoice inv = createAnonInvoice();
    LineItem expItem1 = new LineItem(inv, product, QUANTITY1);
    LineItem expItem2 = new LineItem(inv, product, QUANTITY2);
    inv.addItemQuantity(product, QUANTITY1);
    inv.addItemQuantity(product, QUANTITY2);
    List lineItems = inv.getLineItems();
    assertEquals("number of items", lineItems.size(), 2);
    LineItem actual = (LineItem)lineItems.get(0); // Verify first item
    assertEquals(expItem1.getInv(), actual.getInv());
    assertEquals(expItem1.getProd(), actual.getProd());
    assertEquals(expItem1.getQuantity(), actual.getQuantity());
    actual = (LineItem)lineItems.get(1); // Verify second item
    assertEquals(expItem2.getInv(), actual.getInv());
    assertEquals(expItem2.getProd(), actual.getProd());
    assertEquals(expItem2.getQuantity(), actual.getQuantity());
}
```

33

Test Logic in Production

"The production code contains logic that should be exercised only during tests"

- Test Hook
- For Tests Only
- Test Dependency in Production
- Equality Pollution

34

Hard to Test Code

Code can be hard to test for a number of reasons:

- Too closely coupled to other software
- No interface provided to set state and/or to observe state
- etc.

Root Cause is lack of Design for Testability

- Comes naturally with Test-Driven Development
- Likely to have to be retrofitted to legacy software

Temporary Workaround is Test Hook

- Becomes Test Logic in Production (code smell) if not removed

35

Test Double Pattern

- **Replace depended-on components with test-specific ones to isolate SUT**
- **Kinds of Test Doubles**
  - Test Stubs return test-specific values
  - Test Spies record method calls and arguments for verification by Test Method
  - Mock Objects verify the method calls and arguments themselves
  - Fake Objects provide (apparently) same services in a "lighter" way
- **Test Doubles need to be "installed"**
  - Dependency Injection
  - Dependency Lookup
- **Configurable Test Doubles are reusable but need to be configure with test-specific values**
  - return values
  - expected method calls & arguments

36

Testability Patterns

- **Humble Object**
  - Objects closely coupled to the environment should not do very much (be humble)
  - Should delegate real work to a context-independent testable object
- **Dependency Injection**
  - Client "injects" depended-on objects into SUT
  - Tests can pass a Test Double to control "indirect inputs" from dependents
- **Dependency Lookup**
  - SUT asks another object for it's dependencies
  - Service Locator, Object Factory, Component Registry
- **Test-Specific Subclass**
  - Can extend the SUT to all access by test

37

Behavior Smells

- A problem seen when running tests.
- Tests fail when they should pass
  - or pass when they should fail (rarer)
- The problem is with how tests are coded
  - not a problem in the SUT
- Sniff Test:
  - Detectable via compile or execution behavior of tests

38

General Behavior Smells

- Assertion Roulette
- Erratic Test
- Fragile Test
- Frequent debugging
- Manual Intervention
- Slow Tests

39

Assertion Roulette

- **Symptom:**
  - One or more unit tests are failing in the automated build and you cannot tell why without rerunning the tests in your IDE. When you cannot reproduce the problem in your IDE you have no idea what is going wrong.
- **Impact:**
  - It takes longer to determine what is wrong with the code.
  - Bugs that cannot be reproduced cannot be fixed.
- **Root Cause:**
  - Missing/Unclear Assertion Messages

40

Erratic Test

- Interacting Tests
  - When one test fails, a bunch of other tests fail for no apparent reason because they depend on other tests' side effects
- Unrepeatable Tests
  - Tests can't be run repeatedly without intervention
- Test Run War
  - Seemingly random, transient test failures
  - Only occurs when several people testing simultaneously
- Resource Optimism
  - Tests depend on something in the environment that isn't available
- Non-Deterministic Tests
  - Tests depend on non-deterministic inputs

Fragile Tests

Causes:

- Interface Sensitivity
  - Every time you change the SUT, tests won't compile or start failing
  - You need to modify lots of tests to get things "Green"again
  - Greatly increases the cost of maintaining the system
- Behavior Sensitivity
  - Behavior of the SUT changes but it should not affect test outcome
  - Caused by being dependent on too much of the SUT's behavior.

Fragile Tests

Causes:

- Data Sensitivity
  - Alias: Fragile Fixture
  - Tests start failing when a shared fixture is modified e.g. New records are put into the database
- Context Sensitivity
  - Something outside the SUT changes e.g. System time/date, contents of another application

Frequent Debugging

- Symptom:
  - One or more tests are failing and you cannot tell why without
  - resorting to the debugger. This seems to be happening a lot lately!
- Impact:
  - Debugging is a very time-intensive activity.
  - While it may help you find the bug, it won't keep it from coming back.
- Root Causes:
  - Missing Unit Tests
  - Poor Assertion Messages

Manual Intervention

- Symptom:
  - A test requires a person to perform some manual action each time it is run
- Impact:
  - May result in frequent debugging
  - High test maintenance cost
  - Makes it impractical to have a fully automated Integration build and regression test process
- Causes:
  - Manual Fixture Setup
  - Manual Result Verification
  - Manual Event Injection

Slow Tests

"It takes several minutes to hours to run all the tests"

- Impact
  - Lost productivity caused by waiting for tests
  - Lost quality due to running tests less frequently
- Causes
  - Slow Component Usage e.g. Database
  - Asynchronous Test e.g. Delays or Waits
  - General Fixture e.g. too much fixture being setup

Project Smells

- A Test Smell that a project manager is likely to observe
- Symptoms are typically developer behavior or feedback from other organizations
- There may be metrics that point out the smell
  - e.g. Number of bugs found in Acceptance Test
- Root cause is often Code or Behavior Smells
- Cannot be addressed directly
  - Solution is to address underlying smell's

General Project Smells

- Buggy Tests
- Developers Not Writing Tests
- High Test Maintenance Cost
- Production Bugs

Buggy Tests

- Symptoms:
  - Tests are failing when they shouldn't (the SUT works fine)
- Impact:
  - No one trusts the tests any more
- Possible Causes:
  - Erratic Tests
  - Fragile Tests
  - Untested Test Code

Developer's Not Writing Tests

- **Symptoms:**
  - No tests can be found when you ask to see the
    - unit tests for a task,
    - customer tests for a User Story,
  - Lack of clarity about what a user story or task really means
- **Impact:**
  - Lack of safety net
  - Lack of focus
- **Possible Causes:**
  - Hard to Test Code?
  - Not enough time?
  - Don't have the skills?
  - Have been told not to?
  - Don't see the value?

High Maintenance Costs

- Symptoms:
  - A lot of effort is going into maintaining the tests
- Impact:
  - Cost of building functionality is increasing
  - People are agitating to abandon the automated test
- Possible Causes:
  - Erratic Test
  - Fragile Test
  - Buggy Test
  - Obscure Test
  - Hard to Test Code

Production Bugs

- Symptoms:
  - Bugs are being found in production
- Impact:
  - Expensive trouble-shooting
  - Development team's reputation is in jeopardy
- Possible Causes:
  - Lost/Missing Tests
  - Slow Tests
  - Untested Code
  - Hard-to-Test Code
  - Developers Not Writing Tests

Effective test automation

After test generation by considering all paths and the features and organization specified, our test still may have these bad smells:

- Slow Tests
- Test Code Duplication
- Obscure Tests
- Buggy Tests

so we need Refactoring.