Stunt Doubles For Your Code

By Ed Bartram

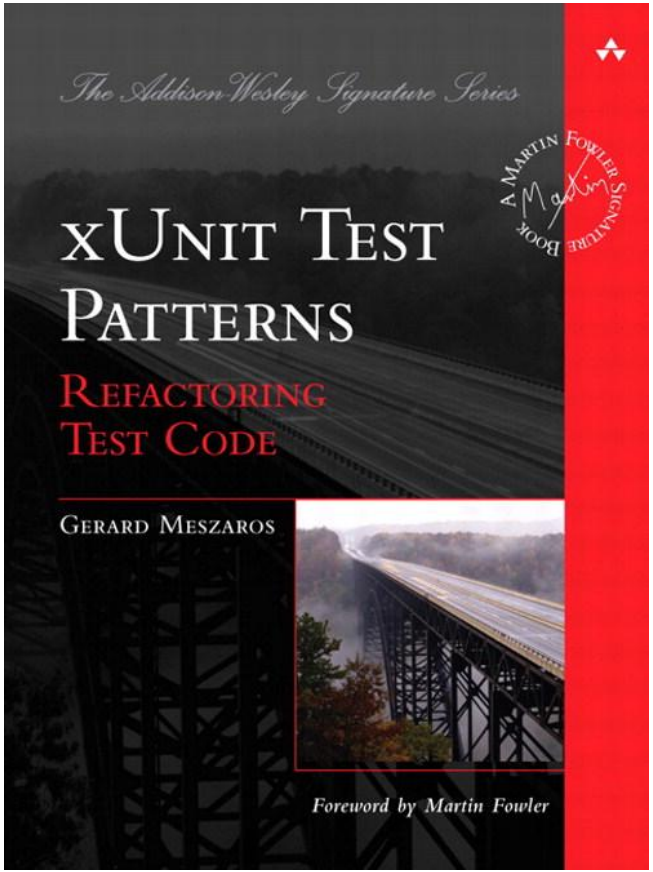
MY HOBBY

TRANSLATING CODING EXAMPLES FROM OTHER LANGUAGES INTO COLDFUSION

XKCF

What am I talking about?

xUnitPatterns.com



Your Code & Hollywood Actors

Stunt Doubles

What is a ...

SUT - System Under Test

What is a ...

DOC - Depended On Component

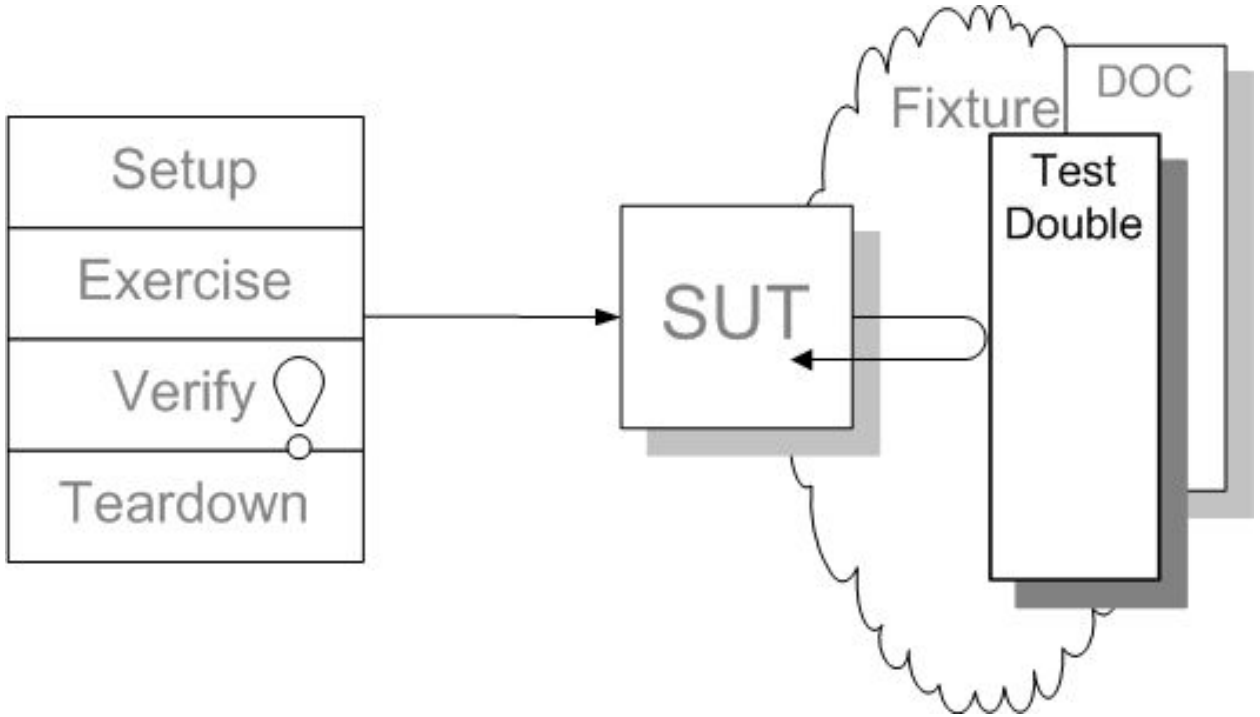
What is ...

Fixture

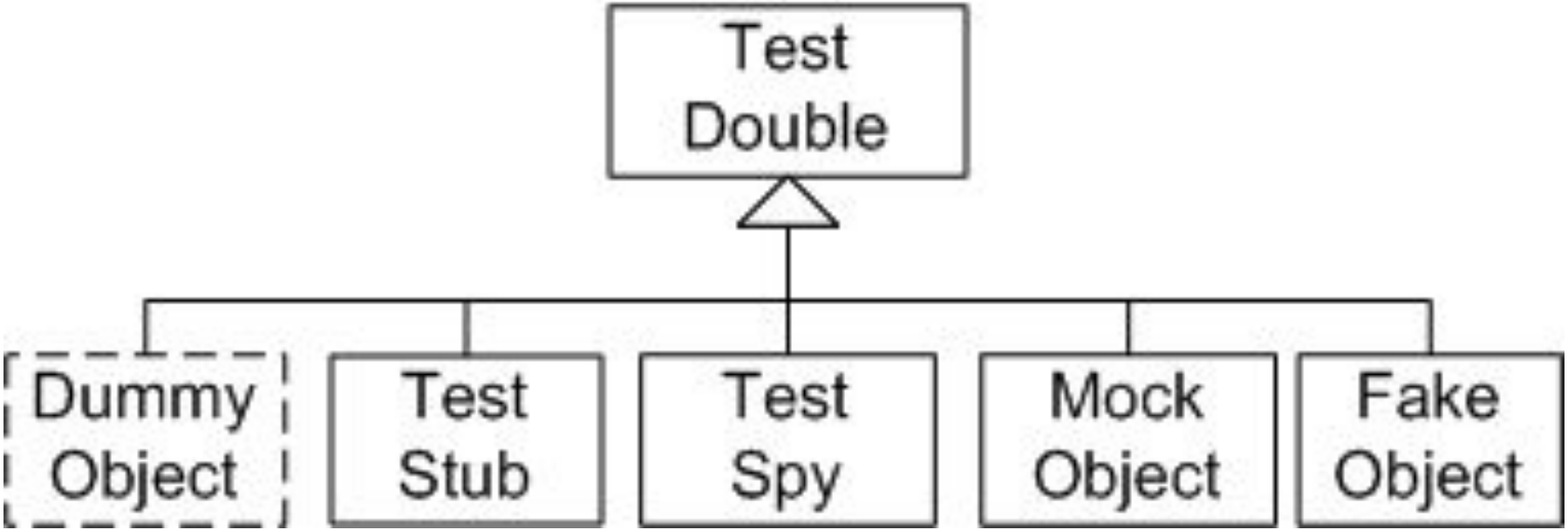
What is ...

Test Double

Putting it all together



Types of Test Doubles



Mock

VS

mock

Dummy

“Methods often take as arguments objects that are stored in instance variables for later use. Often, these objects (or at least some attributes of these objects) are never used in the code that we are actually testing so we are only creating them to conform to the signature of some method we have to call to get the SUT into the right state. Constructing these objects can be non-trivial and adds unnecessary complexity to the test.”

Invoice.cfc

```
public Invoice function init( required iCustomer customer ){
  setCustomer( arguments.customer );
  return this;
}

public void function addItemQuantity( required Product
product, quantity ) {
  setProduct( arguments.product );
  setQuantity( arguments.quantity );
}
```

Customer.cfc

```
component implements="iCustomer" {

  public Customer function init( required Address address ){
    return this;
  }

  public numeric function getZone() {
...
```

Nested Dependencies

```
public Invoice function init( required iCustomer customer ){

public Customer function init( required Address address ) {

public Address function init( …, required City city, … ) {

public City function init( …, required State state ) {

public State function init(string name,string abbreviation){
```

A “Simple” Test Without Doubles

```
public void function testInvoice_addLineItem_noECS() {
  var state = new State( "West Dakota", "WD" );
  var city = new City( "Centreville", state );
  var address = new Address("123 Blake St.", city, "12345");
  var customer = new Customer( ..., address );

  var product = new Product( "Widget", ... );

  var invoice = new Invoice( customer );
  invoice.addItemQuantity( product, 1 );
```

Invoice.cfc

```
public Invoice function init( required iCustomer customer ){
  setCustomer( arguments.customer );
  return this;
}

public void function addItemQuantity( required Product
product, quantity ) {
  setProduct( arguments.product );
  setQuantity( arguments.quantity );
}
```

Replace Unused Values With Dummies

```
public void function testInvoice_addLineItem_DO() {
   var invoice = new Invoice( new CustomerDummy() );
   var product = new Product( "Dummy Product Name", ... );

   invoice.addItemQuantity( product, 1 );

   var lineItems = invoice.getLineItems();
   $assert.isEqual( arrayLen( lineItems ), 1 );
   ...
```

CustomerDummy.cfc

```
component implements="iCustomer" {
  public iCustomer function init() {
    // Real simple; nothing to initialize!
    return this;
  }

  public numeric function getZone() {
    throw( message="This should never be called!" );
  }
```

TestBox Dummy

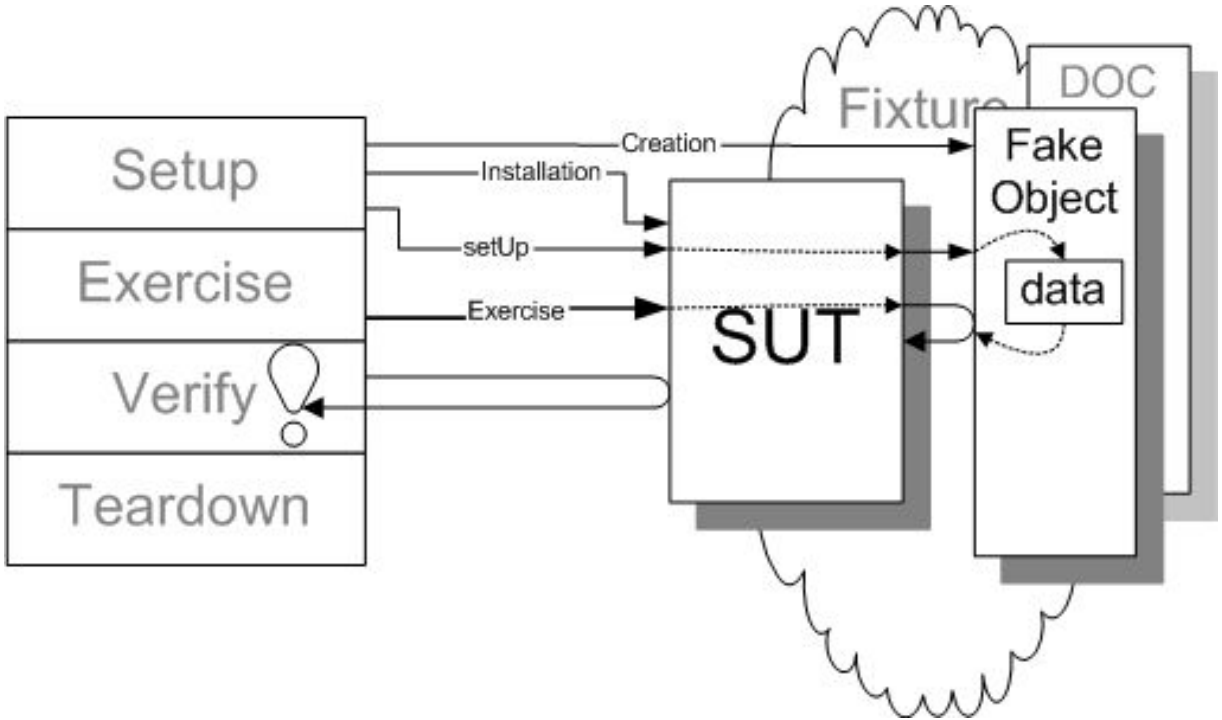
```
var customerDummy = createStub();
customerDummy.$( "getZone" ).$throws( message="This should
never be called!" );

var invoice = new Invoice( customerDummy );

var product = new Product( "Dummy Product Name", … );

invoice.addItemQuantity( product, … );
```

Fake

“The SUT often depend on other components or systems. The interactions with these other components may be necessary but the side-effects of these interactions as implemented by the real depended-on component (DOC), may be unnecessary or even detrimental.

A Fake Object is a much simpler and lighter weight implementation of the functionality provided by the DOC without the side effects we choose to do without.”

Fake


Setup
Exercise
Verify
Teardown
Creation
Installation
setUp
Exercise
Fixture
DOC
Fake Object
data
SUT

FlightManagement.cfc

```
component accessors=true {

  property name="flightDAO";

  public FlightManagement function init() {
    setFlightDAO( new FlightDAO() );
  }
```

FlightManagement.cfc

```
public Airport function createAirport( airportCode, city ) {
  … }

public void function createFlight( origin, destination ) {
  ...
  getFlightDAO().saveFlight( ... );
}

public array function getFlightsByOriginAirport( airport ){
  return getFlightDAO().readFlight( ... );
}
```

Test w/Database Reads & Writes

```
public void function testGetFlights() {
  var flightMgmt = new FlightManagement();
  var YYC = flightMgmt.createAirport( "YYC", "Calgary" );
  var LAX = flightMgmt.createAirport( "LAX", "LA" );
  flightMgmt.createFlight( YYC, LAX );

  var flights = flightMgmt.getFlightsByOriginAirport( YYC );

  $assert.isEqual("YYC", flights[1].getOrigin().getCode() );
```

Replace DAO With a Fake

```
public void function testReadWrite_inMemory() {
  var flightMgmt = new FlightManagement();
  flightMgmt.setFlightDAO( new FlightDAOFake() );
  var YYC = flightMgmt.createAirport( "YYC", "Calgary" );
  var LAX = flightMgmt.createAirport( "LAX", "LA" );
  flightMgmt.createFlight( YYC, LAX );

  var flights = flightMgmt.getFlightsByOriginAirport( YYC );

  $assert.isEqual("YYC", flights[1].getOrigin().getCode() );
```

FlightDAOFake.cfc

```
public void function saveFlight() {}

public array function readFlight( originCode ) {
  var airport = new Airport( "YYC", "Dummy Name" );

  var flight = new Flight();
  flight.setOrigin( airport );

  return [ flight ];
}
```

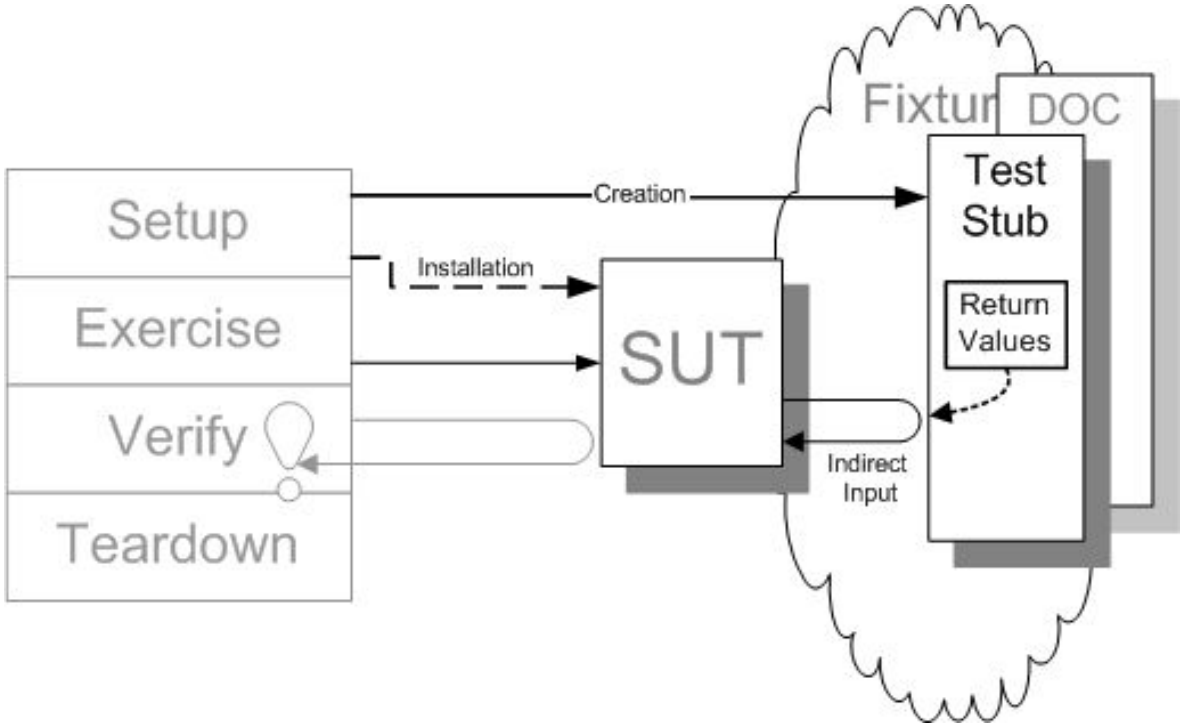
TestBox Fakes

```
airport = querySim( "originCode, city
YYC | Calgary
LAX | Los Angeles
" );

flightDAOFake = createStub();
flightDAOFake.$( "saveFlight" );
flightDAOFake.$( "readFlight", airport );
flightMgmt.setFlightDao( flightDAOFake );
```

Indirect Input

“When the behavior of the system under test (SUT) is affected by the values returned by another component whose services it uses, we call those values indirect inputs of the SUT. Indirect inputs may be actual return values of functions, updated (out) parameters of procedures or subroutines, and any errors or exceptions raised by the depended-on component (DOC). Testing of the SUT behavior with indirect inputs requires the appropriate control point on the "back side" of the SUT.”

Stub


Setup
Exercise
Verify
Teardown
Creation
Installation
SUT
Fixtur
DOC
Test
Stub
Return
Values
Indirect
Input

TimeDisplay.cfc

```
component accessors=true {

  property name="timeProvider";

  public TimeDisplay function init() {
    setTimeProvider( new TimeProvider() );

    return this;
  }
```

TimeDisplay.cfc

```
public string function getCurrentTimeAsHtmlFragment() {
  var time = getTimeProvider().getTime();
  var html = '<span class="tinyBoldText">';

  if ( hour( time ) == 0 && minute( time ) <= 1 ) {
    html &= "Midnight";
  } else if ( hour( time ) == 12 && minute( time ) == 0 ) {
    html &= "Noon";
  } else {
    html &= timeFormat( time, "h:mm tt" );
...
```

A Test Depending on Time

```
public void function testDisplayCurrentTime_AtMidnight() {

  var timeDisplay = new TimeDisplay();

  var actualTime = timeDisplay.getCurrentTimeAsHtml();

  var expectedTime = '<span>Midnight</span>';
  $assert.isEqual( expectedTime, actualTime );
}
```

Replace Time Dependency With a Stub

```
public void function testDisplayCurrentTime_withStub() {
   var timeProviderStub = new TimeProviderStub();
   timeProviderStub.setHours( 0 );
   timeProviderStub.setMinutes( 0 );

   var timeDisplay = new TimeDisplay();
   timeDisplay.setTimeProvider( timeProviderStub );

   var actualTime = timeDisplay.getCurrentTimeAsHtml();
   $assert.isEqual( expectedTime, actualTime );
```

TimeProviderStub.cfc

```
public string function setHours( hours ) {
 setMyTime(hours, minute(getMyTime()), second(getMyTime()));
}
public string function setMinutes( minutes ) {
 setMyTime(hour(getMyTime()), minutes, second(getMyTime()));
}
public void function setMyTime( hours, minutes ) {
 variables.myTime = createTime( hours, minutes, 0 );
}
```

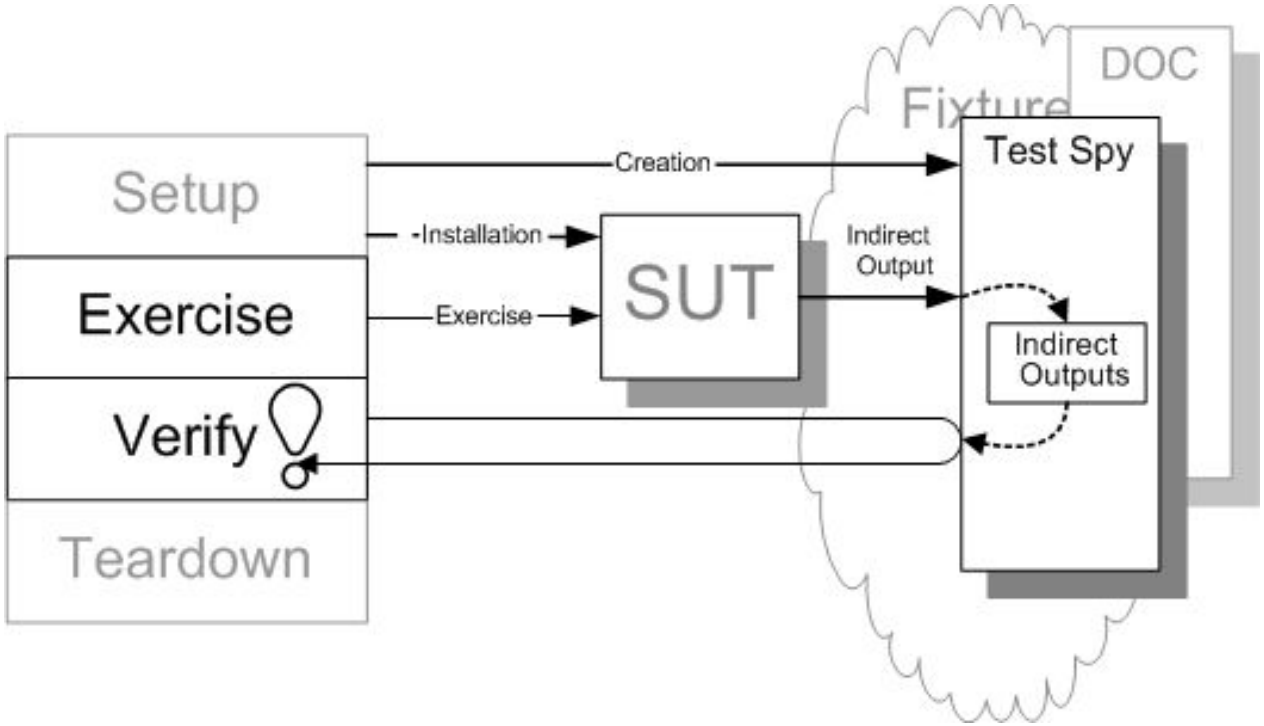
TestBox Stub

```
timeProviderStub = createStub( extends="TimeProvider" );
timeProviderStub.init();

timeProviderStub.$property(propertyName="hours", mock=3);
timeProviderStub.$property(propertyName="minutes", mock=14);
timeProviderStub.$( method="getHourOfDay", returns=25 );

$assert.isEqual( 3, timeProviderStub.getHours() );
$assert.isEqual( 14, timeProviderStub.getMinutes() );
$assert.isEqual( 25, timeProviderStub.getHourOfDay() );
```

Indirect Output

“When the behavior of the system under test (SUT) includes actions that cannot be observed through the public API of the SUT but which are seen or experienced by other systems or application components, we call those actions the indirect outputs of the SUT. Indirect outputs may be method or function calls to another component, messages sent on a message channel (e.g. MQ or JMS), records inserted into a database or written to a file.”

Spy


DOC
Fixture
Test Spy
Setup
Creation
Installation
SUT
Indirect
Output
Exercise
Exercise
Indirect
Outputs
Verify
Teardown

FlightManagement.cfc removeFlight()

```
property name="auditLog";

public FlightManagement function init() {
   setAuditLog( new AuditLog() );
… }

public void function removeFlight( string flightNumber ) {
   getAuditLog().logMessage( dateFormat( now() ), "Ed", "RF",
arguments.flightNumber );
… }
```

Audit Log.cfc

```
component accessors=true {

    property name="user" type="string";
    property name="actionCode" type="string";
    property name="detail" type="string";

    public any function init() { … }

    public void function logMessage( … ) { … }
}
```

AuditLogSpy.cfc - init()

```
component accessors=true {
  property name="user" type="string";
  property name="actionCode" type="string";
  property name="detail" type="string";

  property name="numberOfCalls" type="numeric";

  public any function init() {
    setNumberOfCalls( 0 );
    ... }
```

AuditLogSpy.cfc - logMessage()

```
public void function logMessage( user, actionCode, detail ){
  setUser( arguments.user );
  setActionCode( arguments.actionCode );
  setDetail( arguments.detail );

  variables.numberOfCalls++;
}
```

Test Spy Installation

```
public void function testRemoveFlightLogging_recordingSpy(){
  var flightMgmt = new FlightManagement();

  var auditLogSpy = new AuditLogSpy();
  flightMgmt.setAuditLog( auditLogSpy );

  flightMgmt.removeFlight( expected.getFlightNumber() );
```

Replace With A Spy - Assertions

```
$assert.isFalse( flightMgmt.flightExists( … ) );

$assert.isEqual( "RF", auditLogSpy.getActionCode() );
$assert.isEqual( "Ed", auditLogSpy.getUser() );
$assert.isEqual( expected.getFlightNumber(),
auditLogSpy.getDetail() );

$assert.isEqual( 1, auditLogSpy.getNumberOfCalls() );
```

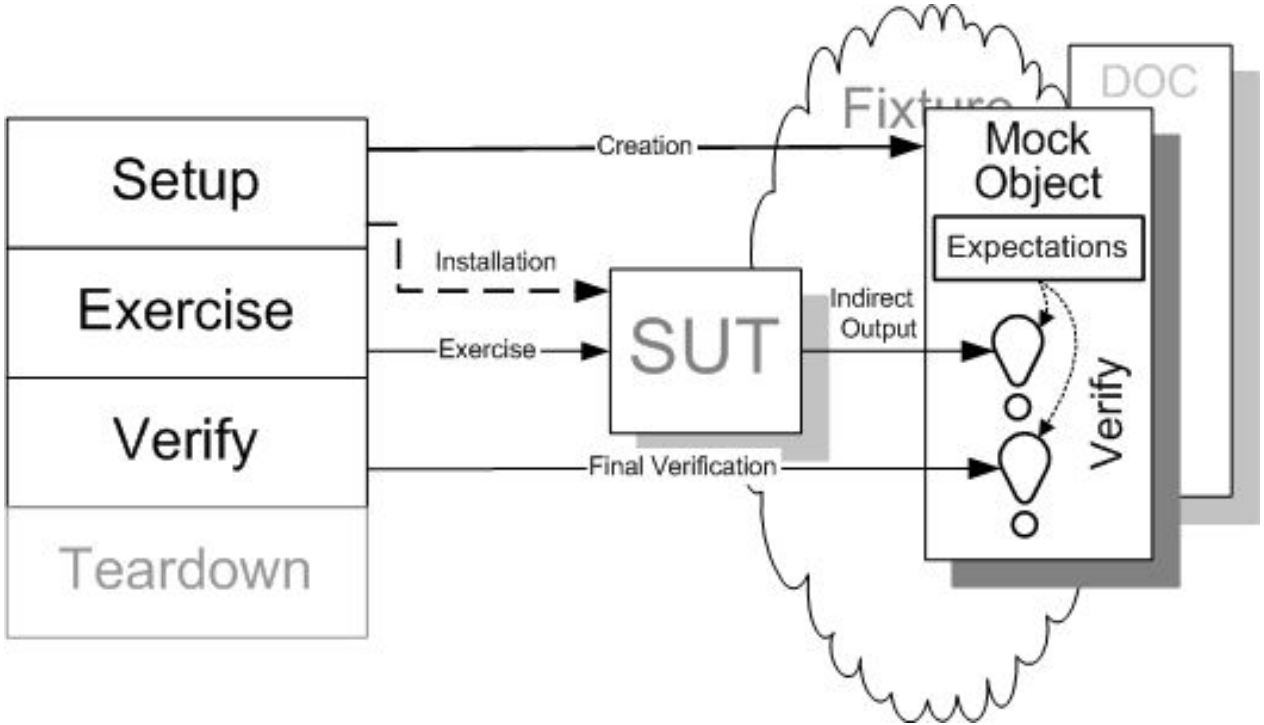
TestBox Spy

```
var auditLogStub = createStub( extends="AuditLog" );
auditLogStub.$( "logMessage" );

var flightMgmt = new TestDoubles.FlightManagement();
flightMgmt.setAuditLog( auditLogStub );
flightMgmt.removeFlight( "Dummy Flight Number" );

var numberOfCalls = auditLogStub.$count( "logMessage" );
$assert.isEqual( 1, numberOfCalls );
```

TestBox Spy Sort Of...

```
public function testSpyMakePublic() {
  var flightMgmt = new FlightManagement();
  prepareMock( flightMgmt );
  makePublic( flightMgmt, "myPrivateMethod" );

  var results = flightMgmt.myPrivateMethod();

  $assert.isNotEmpty( results );
}
```

Mock


Setup
Exercise
Verify
Teardown
Creation
Installation
Exercise
Final Verification
SUT
Indirect Output
Fixture
DOC
Mock Object
Expectations
Verify

FlightManagement.cfc removeFlight()

```
property name="auditLog";

public FlightManagement function init() {
   setAuditLog( new AuditLog() );
… }

public void function removeFlight( string flightNumber ) {
   getAuditLog().logMessage( dateFormat( now() ), "Ed", "RF",
arguments.flightNumber );
… }
```

AuditLogMock.cfc

```
component accessors=true extends="testbox.system.BaseSpec" {
    property name="user" type="string";
    property name="actionCode" type="string";
    property name="detail" type="string";
    property name="expectedUser" type="string";
    property name="expectedActionCode" type="string";
    property name="expectedDetail" type="string";
    property name="expectedNumberCalls" type="numeric";
    property name="actualNumberCalls" type="numeric";
    property name="logMessageCalled" type="boolean";
```

AuditLogMock.cfc - init()

```
public any function init() {
   setActualNumberCalls( 0 );

   setLogMessageCalled( false );

   return this;
}
```

AuditLogMock.cfc - setExpectedLogMessage()

```
public void function setExpectedLogMessage(
expectedUser, expectedActionCode, expectedDetail ) {

  setExpectedUser( arguments.expectedUser );
  setExpectedActionCode( arguments.expectedActionCode );
  setExpectedDetail( arguments.expectedDetail );
}
```

AuditLogMock.cfc - logMessage()

```
public void function logMessage( user, actionCode, detail) {
   setLogMessageCalled( true );
   variables.actualNumberCalls++;

   $assert.isEqual( getExpectedUser(), user );
   $assert.isEqual( getExpectedActionCode(), actionCode );
   $assert.isEqual( getExpectedDetail(), detail );
   $assert.isEqual(getExpectedNumberCalls(),actualNumberCalls);
}
```

AuditLogMock.cfc - verify()

```
public void function verify() {
   $assert.isTrue( getLogMessageCalled() );
}
```

Test - Mock Configuration

```
public void function testRemoveFlight_Mock() {
  var auditLogMock = new AuditLogMock();

  // mock configuration
  auditLogMock.setExpectedLogMessage( "Dummy User", "Dummy Action Code", "Dummy Flight Number" );

  auditLogMock.setExpectedNumberCalls( 1 );
```

Test - Mock Installation

```
// mock installation
var flightMgmt = new FlightManagement();
flightMgmt.setAuditLog( auditLogMock );

// exercise
flightMgmt.removeFlight( getFlightNumber() );
```

Replace With a Mock (Installation)

```
  // verify
  $assert.isFalse( flightMgmt.flightExists( … ) );

  auditLogMock.verify();
}
```

TestBox Mock?

Summary

Please use the correct terminology when possible

- **Test Doubles** - replace dependencies used by code you want to test
- **Dummies** - values we don't care about, but are required minimally
- **Fakes** - lightweight implementations of external dependencies
- **Stubs** - are like Dummies, but with **Indirect Inputs**, injected values that can control the flow of logic within our SUT
- **Spies** - are like Stubs, but provide observation points to **Indirect Outputs**, unobservable behaviors within our SUT
- **Mocks** - are like Spies, but instead provide verification to **Indirect Outputs**, unobservable behaviors within our SUT

And ...

Use Test Doubles to

- isolate the code you are testing from its dependencies.
- simplify complicated setups with **Dummies**
- remove difficult or slow external connections from your test with **Fakes**
- control the logic within your SUT using **Stubs**
- provide observation points or verification to unobservable behaviors in your SUT with **Spies** and **Mocks.**

Resources

- xUnit Test Patterns by Gerard Meszaros
  http://xunitpatterns.com/
- TestBox & MockBox Manual v2.x
  https://testbox.ortusbooks.com/
- Effective Unit Testing with Test Doubles by Matt Logan
  https://www.youtube.com/watch?v=_pCwcdNtxog
- Mocks, Stubs, and Spies, Oh My! by Brian Gardner
  https://www.youtube.com/watch?v=tVCSKsMtXn0
- Mocks Aren't Stubs, Fakes, Dummies, or Spies by Dave Marshall
  https://www.youtube.com/watch?v=6_r3AzRg1HM
- Spaceballs on IMDB
  https://www.imdb.com/title/tt0094012/?ref_=nv_sr_1
- Randall Munroe's XKCD
  https://xkcd.com/

Thank You!

Questions / Feedback

Contact Me

Website: edbartram.com | edbartram.github.io
GitHub: edbartram
Slack: @edbartram
Twitter: @edbartram
LinkedIn: in/edbartram