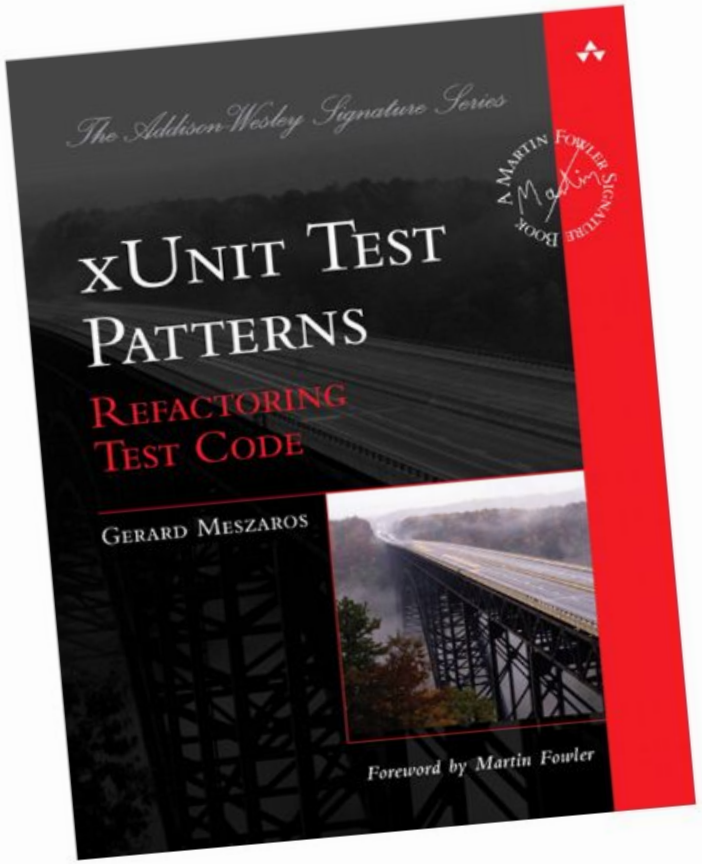
Testing



http://xunitpatterns.com/

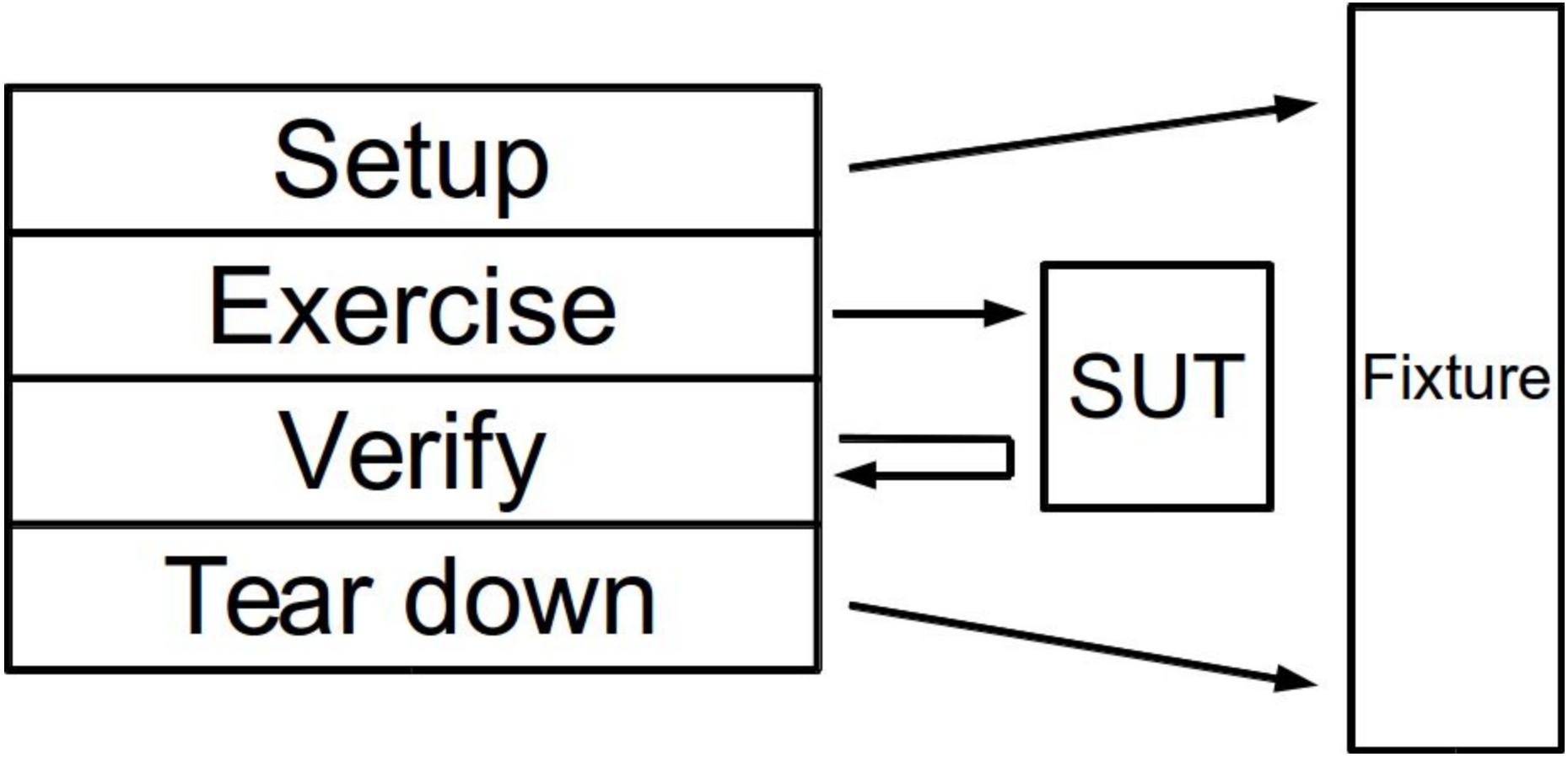
Valores/Beneficios

Calidad

Entendimiento

Riesgos

Test Automático 101


Setup
Exercise
Verify
Tear down
SUT
Fixture

Caracteristicas deseadas

Facil de correr

Facil de Escribir y Mantener

Mantenimiento mínimo

Principios

Design For Testability

Use the Front Door First

Communicate Intent

Keep Test's Independent

Insolate the SUT

Minimize Test Overlap

Keep Test Logic
out of Production Code

Verify One Condition per Test

Test Smells

Eager Test

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

Mystery Guest

```
    public void testGetFlightsByFromAirport_OneOutboundFlight_mg() throws Exception {
        loadAirportsAndFlightsFromFile("test-flights.csv");
        // Exercise System
        List flightsAtOrigin = facade.getFlightsByOriginAirportCode("YYC");
        // Verify Outcome
        assertEquals(1, flightsAtOrigin.size());
        FlightDto firstFlight = (FlightDto) flightsAtOrigin.get(0);
        assertEquals("Calgary", firstFlight.getOriginCity());
    }
```

General Fixture

```
    public void testGetFlightsByFromAirport_OneOutboundFlight() throws Exception {
        setupStandardAirportsAndFlights();
        FlightDto outboundFlight = findOneOutboundFlight();
        // Exercise System
        List flightsAtOrigin = facade.getFlightsByOriginAirport(
                outboundFlight.getOriginAirportId());
        // Verify Outcome
        assertOnly1FlightInDtoList("Flights at origin", outboundFlight,
                flightsAtOrigin);
    }

    public void testGetFlightsByFromAirport_TwoOutboundFlights() throws Exception {
        setupStandardAirportsAndFlights();
        FlightDto[] outboundFlights = findTwoOutboundFlightsFromOneAirport();
        // Exercise System
        List flightsAtOrigin = facade.getFlightsByOriginAirport(
                outboundFlights[0].getOriginAirportId());
        // Verify Outcome
        assertExactly2FlightsInDtoList("Flights at origin", outboundFlights,
                flightsAtOrigin);
    }
```

Irrelevant Information

```
  public void testGetFlightsByOriginAirport_TwoOutboundFlights() throws Exception {
        FlightDto expectedCalgaryToSanFran = new FlightDto();
        expectedCalgaryToSanFran.setOriginAirportId(calgaryAirportId);
        expectedCalgaryToSanFran.setOriginCity(CALGARY_CITY);
        expectedCalgaryToSanFran.setDestinationAirportId(sanFranAirportId);
        expectedCalgaryToSanFran.setDestinationCity(SAN_FRAN_CITY);
        expectedCalgaryToSanFran.setFlightNumber(
                facade.createFlight(calgaryAirportId, sanFranAirportId));
        FlightDto expectedCalgaryToVan = new FlightDto();
        expectedCalgaryToVan.setOriginAirportId(calgaryAirportId);
        expectedCalgaryToVan.setOriginCity(CALGARY_CITY);
        expectedCalgaryToVan.setDestinationAirportId(vancouverAirportId);
        expectedCalgaryToVan.setDestinationCity(VANCOUVER_CITY);
        expectedCalgaryToVan.setFlightNumber(facade.createFlight(
                calgaryAirportId, vancouverAirportId));

        List lineItems = inv.getLineItems();
        assertEquals("number of items", lineItems.size(), 2);
        //   verify first item
        LineItem actual = (LineItem) lineItems.get(0);
        assertEquals(expItem1.getInv(), actual.getInv());
        assertEquals(expItem1.getProd(), actual.getProd());
        assertEquals(expItem1.getQuantity(), actual.getQuantity());
        //   verify second item
        actual = (LineItem) lineItems.get(1);
        assertEquals(expItem2.getInv(), actual.getInv());
        assertEquals(expItem2.getProd(), actual.getProd());
        assertEquals(expItem2.getQuantity(), actual.getQuantity());
    }
```

Hard-Coded Test Data

```
    @Test
    public void testPagarConPuntos() {
        Producto VinoXYZ = new Producto("VinoXYZ", CategoriaProducto.Bebidas, 14);
        Producto Coca = new Producto("Coca", CategoriaProducto.Bebidas, 1);

        Venta venta = new Venta(Dia.Lunes);
        ventas.setPuntaje(12);
        ventas.agregarProducto(VinoXYZ, 1);
        ventas.agregarProducto(Coca, 2);

        double totalObtenido = venta.calcularTotal();
        assertEquals(16.0, totalObtenido, 0.1);

        totalObtenido = venta.calcularTotal(12);
        assertEquals(4.0, totalObtenido, 0.1);

        assertEquals(0, venta.getTotalPuntaje());
    }
```

Indirect Testing

```
public void testAnalyze_sameAirline_LessThanConnectionLimit() throws Exception {
        // setup
        FlightConnection illegalConn = createSameAirlineConn(LEGAL_CONN_MINS_SAME - 1);
        // exercise
        FlightConnectionAnalyzerImpl sut = new FlightConnectionAnalyzerImpl();
        String actualHtml = sut.getFlightConnectionAsHtmlFragment(illegalConn.
getInboundFlightNumber(), illegalConn.getOutboundFlightNumber());
        // verification
        StringBuilder expected = new StringBuilder();
        expected.append("<span class='boldRedText'>");
        expected.append("Connection time between flight ");
        expected.append(illegalConn.getInboundFlightNumber());
        expected.append(" and flight ");
        expected.append(illegalConn.getOutboundFlightNumber());
        expected.append(" is ");
        expected.append(illegalConn.getActualConnectionTime());
        expected.append(" minutes.</span>");
        assertEquals("html", expected.toString(), actualHtml);
    }
```

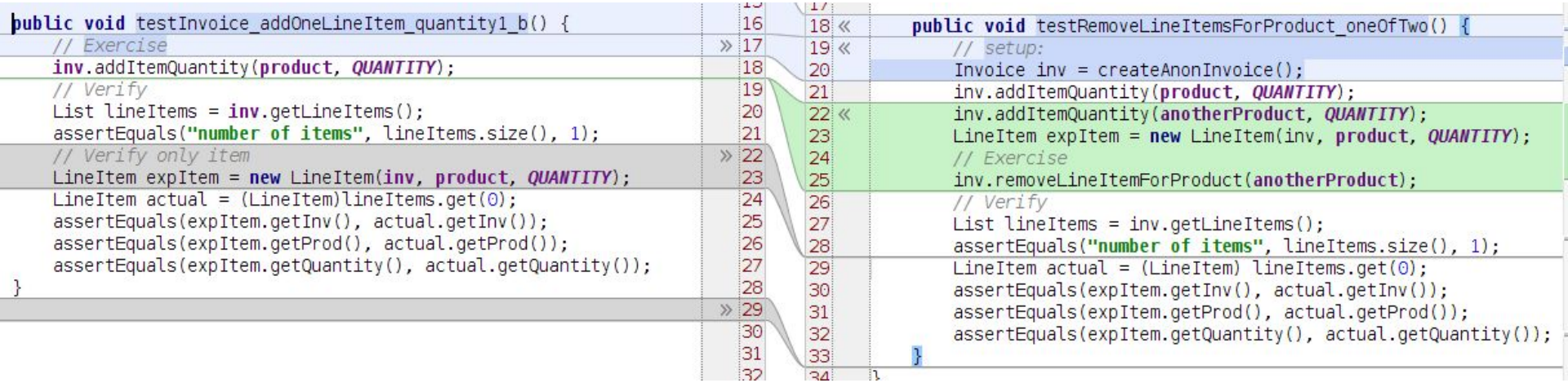
Flexible Test

```
public void testDisplayCurrentTime_whenever() {
        // fixture setup
        TimeDisplay sut = new TimeDisplay();
        // exercise sut
        String result = sut.getCurrentTimeAsHtmlFragment();
        // verify outcome
        Calendar time = new DefaultTimeProvider().getTime();
        StringBuilder expectedTime = new StringBuilder();
        expectedTime.append("<span class='tinyBoldText'>");
        if ((time.get(Calendar.HOUR_OF_DAY) == 0) && (time.get(Calendar.MINUTE) <= 1)) {
            expectedTime.append("Midnight");
        } else if ((time.get(Calendar.HOUR_OF_DAY) == 12)
                && (time.get(Calendar.MINUTE) == 0)) { // noon
            expectedTime.append("Noon");
        } else {
            SimpleDateFormat fr = new SimpleDateFormat("h:mm a");
            expectedTime.append(fr.format(time.getTime()));
        }
        expectedTime.append("</span>");
        assertEquals(expectedTime.toString(), result);
    }
```

Conditional Verification Logic

```
//verify Vancouver is in the list:
actual = null;
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;
        }
}
```

Test Code Duplication



```
public void testInvoice_addOneLineItem_quantity1_b() {
    // Exercise
    inv.addItemQuantity(product, QUANTITY);
    // Verify
    List lineItems = inv.getLineItems();
    assertEquals("number of items", lineItems.size(), 1);
    // Verify only item
    LineItem expItem = new LineItem(inv, product, QUANTITY);
    LineItem actual = (LineItem)lineItems.get(0);
    assertEquals(expItem.getInv(), actual.getInv());
    assertEquals(expItem.getProd(), actual.getProd());
    assertEquals(expItem.getQuantity(), actual.getQuantity());
}
```

```
    public void testRemoveLineItemsForProduct_oneOfTwo() {
        // setup:
        Invoice inv = createAnonInvoice();
        inv.addItemQuantity(product, QUANTITY);
        inv.addItemQuantity(anotherProduct, QUANTITY);
        LineItem expItem = new LineItem(inv, product, QUANTITY);
        // Exercise
        inv.removeLineItemForProduct(anotherProduct);
        // Verify
        List lineItems = inv.getLineItems();
        assertEquals("number of items", lineItems.size(), 1);
        LineItem actual = (LineItem) lineItems.get(0);
        assertEquals(expItem.getInv(), actual.getInv());
        assertEquals(expItem.getProd(), actual.getProd());
        assertEquals(expItem.getQuantity(), actual.getQuantity());
    }
```

¿Preguntas?