# Test Automation .NET

## Diferentes Tipos de Tests

¿Entonces de qué tipos de pruebas automatizadas vamos a tratar exactamente?, Para establecer el marco sobre el cuál trabajaríamos vamos a ayudarnos de esta matriz.

Esta matriz cataloga diferentes tipos de testing, especialmente dentro de un contexto Agile, nos muestra los diferentes contextos que debemos tener en cuenta para probar adecuadamente nuestro producto.

Los tests dentro de esta matriz han sido categorizados respondiendo a 2 preguntas:

* ¿La prueba está más orientada al negocio o a la tecnología?
* ¿La prueba da soporte al trabajo del equipo o sirven para criticar las características del producto?
* *Orientadas al Negocio:* Son aquellas pruebas expresadas en un lenguaje que es perfectamente entendido por una persona de negocio. Son mejores definidas por BA, PO. Involucran la participación del cliente. Ejm: Sí se retira más dinero del que se tiene en una cuenta, se extenderá automáticamente un préstamo por la diferencia.
* *Orientadas a la Tecnología:* Expresadas en un lenguaje que es bien entendido por alguien relacionado al ámbito tecnológico. Se definen mejor por Developers Testers, DBA, etc. Ejm: La aplicación debe soportar IE7 y EI8.
* *Soporte al Equipo:* Son aquellas que definen qué es lo que el software debe hacer. Ejm: Si llamamos al método Suma con 2 y 2, debe retornar 4; Sí hacemos click en el link “Detalles de la cuenta” debemos ver los movimientos realizados.
* *Critican el Producto:* Son aquellos que tratan de identificar y evitar errores en el producto terminado. Ejm: Cuando me logeo con Brian, ve la información de Tom; Cuando hay 1000 usuarios concurrentes, el tiempo de respuesta se incrementa en 10 segundos.

**CUADRANTES**

* Q1: Ayudan al equipo en la búsqueda y prevención de defectos (Internal Quality) dentro del producto que se está trabajando. Se utilizan apara asegurar que el código escrito hace lo que se espera que haga.
* Q2: Guían el desarrollo realizado por el equipo a través de prueba orientadas al negocio. Estas pruebas permiten comprender y descubrir mejor el negocio las necesidades del cliente. (External Quality / Done Definition)
  + Pruebas de Aceptación: Escenarios o ejemplos expresados en tests ejecutables.
  + Prototipos / Diagramas / Mapas mentales.
* Q3: Evaluan el producto recreando la experiencia actual del usuario. Realizando pruebas realistas del sistema.
  + Demos:
    - Iteration Reviews (formal)
    - Exploratoring Testing con el cliente (informal)
  + Usability Testing.
  + User Acceptance Tests (UAT): Beta teststing.
* Q4: Evaluan el producto, principalmente en términos de atributos de calidad que son particularmente necesarios de cumplir para el software en cuestión.
  + Scalabilty / Compatibility / Security
  + Databa Migration / Memory Managment / Recovery

# Unit Testing

## Analogía del Automóvil

Para entender mejor a que nos referimos cuando hablamos de unit testing. En vez de hablar por el momento de software, vamos a imaginarnos la construcción de un carro.

Un carro tiene un motor, una transmisión, un sistema de frenos, refrigeración, cinturones de seguridad, faros, etc. Un automóvil es la suma de todas esas partes o subsistemas. Cada una de estas partes tienes sus propias especificaciones y requisitos que definen la funcionalidad esperada, tolerancias e interfaces con el resto del sistema.

Imaginen el punto en el cuál todos esos subsistemas son ensamblados y atornillados en una línea de montaje sin que se realice ninguna prueba previa. El conductor que va a realizar la prueba del auto se sienta y este no arranca. ¿ De que subsistema fue la culpa? Si no se realizaron pruebas independientes a cada uno de esos subsistemas y sus partes será muy difícil encontrar el problema.

Suponiendo que se pone en marcha y avanza ¿el piloto podrá probar cada una de las partes del auto? Que el sistema de refrigeración funcione a altas revoluciones o que el motor funcione si el aceite está más abajo que los niveles permisibles. Para realizar todas estas pruebas el piloto se tomará una gran cantidad de tiempo y también serían mejor realizadas por las personas especializadas que construyeron el subsistema y de una manera que el usuario final sería incapaz de hacerlo.

Lo mismo ocurre con el software, de manera similar a los autos, el software es una colección de componentes y partes. Realizar pruebas independientes a cada una de estas partes nos ayudará a depurar el API antes de que alguien intente usarla. Si a través de alguna de estas pruebas falla será muchísimo más fácil de encontrar y corregir el error en comparación a si estuviéramos probando todo el sistema, asimismo se permite ahorrar mucho más tiempo y dinero ya que es más eficiente para cualquier corregir errores antes de entrar a una fase de ensamblaje y detener a todo el equipo ante cada fallo.

Piensen en esto, ustedes preferirían conducir un automóvil cuyos frenos han sido alta y consecutivamente probados por los ingenieros especializados o confiarían en las pruebas de sistema que ha realizado el piloto de pruebas. La misma lógica debemos seguir al desarrollar un software, las pruebas no es algo que se deba realizar luego de que todas las partes han sido combinadas sino se debe realizar de manera independiente a cada parte antes de ensamblarlas en un sistema mucho más grande. Realizar estas pruebas desde el inicio y ejecutarlas constantemente cada vez que se realice algún cambio nos ayudará a a garantizar la calidad del software que las usar y a identificar problemas antes de llegar a QA o a los usuarios finales.

# Test Doubles

## Inversión de Dependencia

public interface IDataAccess  
{  
    decimal GetShippingCosts(Order order);  
  
    void SaveOrder(Order order);  
  
    Order GetOrder(int id);  
}

private IDataAccess dataAccess;  
public OrderServices()  
{  
    this.dataAccess = new DataAccess();  
}

## Inyección de Dependencias

[TestClass]  
public class OrderServicesTests  
{  
    [TestMethod]  
    public void CalculateTotal\_OrderFromUS\_FreeShipping()  
    {  
 Order order = new Order { Country = "US", ItemTotal = 100 };  
        OrderServices orderServices = new OrderServices(new DataAccess());  
  
        var total = orderServices.CalculateTotal(order);  
  
        Assert.AreEqual(100, total);  
    }  
  
    [TestMethod]  
    public void CalculateTotal\_OrderOutsideUS\_ShippingCostIsAdded()  
    {  
 Order order = new Order { Country = "PER", ItemTotal = 100 };  
        OrderServices orderServices = new OrderServices(new DataAccess());  
  
        var total = orderServices.CalculateTotal(order);  
  
        Assert.AreEqual(110, total);  
    }  
  
    [TestMethod]  
    public void Save\_ValidOrder\_TheOrderIsPersisted()  
    {  
 Order order = new Order { Id = 1, Country = "PER", ItemTotal = 100, Total = 110 };  
        OrderServices orderProcessor = new OrderServices(new DataAccess());  
  
        orderProcessor.Save(order);  
  
        Order orderFromDb = orderProcessor.GetOrder(order.Id);  
        Assert.IsNotNull(orderFromDb);  
    }  
}

## Test Doubles Manuales

    [TestClass]  
    public class OrderServicesTests\_ManualTestDoubles  
    {  
        [TestMethod]  
        public void CalculateTotal\_OrderFromUS\_FreeShipping()  
        {  
            OrderServices orderServices = new OrderServices(null);  
            Order order = new Order { Country = "US", ItemTotal = 100 };  
   
            var total = orderServices.CalculateTotal(order);  
   
            Assert.AreEqual(100, total);  
        }  
   
        [TestMethod]  
        public void CalculateTotal\_OrderOutsideUS\_ShippingCostIsAdded()  
        {  
            Order order = new Order { Country = "PER", ItemTotal = 100 };  
            var dataAccess = new SimpleDataAccess();  
            dataAccess.ShippingCosts = 10;  
            OrderServices orderServices = new OrderServices(dataAccess);  
   
            var total = orderServices.CalculateTotal(order);  
   
            Assert.AreEqual(110, total);  
        }  
   
        [TestMethod]  
        public void Save\_ValidOrder\_TheOrderIsPersisted()  
        {  
            Order order = new Order { Id = 1, Country = "PER", ItemTotal = 100, Total = 110 };  
            var dataAccess = new SimpleDataAccess();  
            OrderServices orderProcessor = new OrderServices(dataAccess);  
   
            orderProcessor.Save(order);  
   
            Assert.AreEqual(order,dataAccess.OrderSaved);  
        }  
   
        private class SimpleDataAccess : IDataAccess  
        {  
            public decimal ShippingCosts;  
            public decimal GetShippingCosts(Order order)  
            {  
                return this.ShippingCosts;  
            }  
   
            public Order GetOrder(int id)  
            {  
                throw new **NotImplementedException**();  
            }  
   
            public Order OrderSaved;  
            public void SaveOrder(Order order)  
            {  
                OrderSaved = order;  
            }  
        }  
    }

## Test Double Framework

[TestClass]  
public class OrderServicesTests  
{  
    [TestMethod]  
    public void CalculateTotal\_OrderFromUS\_FreeShipping()  
    {  
        Order order = new Order { Country = "US", ItemTotal = 100 };  
        OrderServices orderServices = new OrderServices(null);  
  
        var total = orderServices.CalculateTotal(order);  
  
        Assert.AreEqual(100, total);  
    }  
  
    [TestMethod]  
    public void CalculateTotal\_OrderOutsideUS\_ShippingCostIsAdded()  
    {  
        Order order = new Order { Country = "PER", ItemTotal = 100 };  
        var dataAccess = new Mock<IDataAccess>();  
        dataAccess.Setup(x => x.GetShippingCosts(order)).Returns(10);  
        OrderServices orderServices = new OrderServices(dataAccess.Object);  
  
        var total = orderServices.CalculateTotal(order);  
  
        Assert.AreEqual(110, total);  
    }  
  
    [TestMethod]  
    public void Save\_ValidOrder\_TheOrderIsPersisted()  
    {  
        Order order = new Order { Id = 1, Country = "PER", ItemTotal = 100, Total = 110 };  
        var dataAccess = new Mock<IDataAccess>();  
        OrderServices orderProcessor = new OrderServices(dataAccess.Object);  
  
        orderProcessor.Save(order);  
  
        dataAccess.Verify(x => x.SaveOrder(order));  
    }  
}

## Mocking Frameworks

Si se encuentra antes o igual en el índice de la lista, es válido

    [TestClass]  
    public class LogManagerTests  
    {  
        private Mock<IConfiguration> configuration;  
        private Mock<IEmailSender> emailSender;  
        private Mock<IAppender> appender;  
        private LogManager logManager;  
   
        [TestInitialize]  
        public void Setup()  
        {  
            this.configuration = new Mock<IConfiguration>();  
            this.emailSender = new Mock<IEmailSender>();  
            this.appender = new Mock<IAppender>();  
            this.logManager = new LogManager(configuration.Object, emailSender.Object, appender.Object);  
        }  
   
        [TestMethod]  
        public void IsEnabled\_MessageLevelBeforeLoggerLevel\_ReturnTrue()  
        {  
            configuration.Setup(x => x.LoggerLevel()).Returns(Level.Info);  
   
            bool isEnabled = this.logManager.IsEnabled(Level.Error);  
   
            Assert.IsTrue(isEnabled);  
        }  
   
        [TestMethod]  
        public void IsEnabled\_MessageLevelAfterLoggerLevel\_ReturnFalse()  
        {  
            configuration.Setup(x => x.LoggerLevel()).Returns(Level.Info);              
   
            bool isEnabled = logManager.IsEnabled(Level.Debug);  
   
            Assert.IsFalse(isEnabled);  
        }  
   
        [TestMethod]  
        public void Write\_LevelError\_SendMailtoAdmin()  
        {  
            logManager.Write("message", Level.Error);  
   
            emailSender.Verify(x => x.SendToAdmin("message"));  
        }  
   
        [TestMethod]  
        public void Write\_IsEnabled\_WriteToAppender()  
        {  
            configuration.Setup(x => x.LoggerLevel()).Returns(Level.Info);  
   
            logManager.Write("message", Level.Info);  
   
            appender.Verify(x => x.Write("message"));  
        }  
    }