

FACULTAD DE INGENIERÍA

ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA DE SISTEMAS

**INFORME PROGRAMACION ORIENTADA A OBJETOS**

**“Pruebas Unitarias”**

**Autor(es):**

* *Celestino Huamán, Angie*
* *Díaz Ocampo, Ricardo*
* *Flores Santisteban, José*
* *Muñoz Ccanto, Joel*
* *Pinto Carhuamaca, Jeremy*

**Asesor:**

*Ing. Coronel Castillo, Gustavo*

**Lima – Perú**

2017

**Introducción:**

Todos los programadores saben que deben realizar pruebas a su código, pocos lo hacen de manera proactiva, la respuesta generalizada al “¿por qué no?” es “no tengo tiempo”.

Este apuro se convierte en un círculo vicioso. Cuanta más presión se siente, menos pruebas se realizan. Cuantas menos pruebas se realizan, menos productivo se es y el código se vuelve menos estable. Cuanto menos productivo y preciso se es, más presión se siente.

**¿Que son las pruebas unitarias?**

En [programación](https://es.wikipedia.org/wiki/Programaci%C3%B3n), una prueba unitaria es una forma de comprobar el correcto funcionamiento de una unidad de código. Por ejemplo en [diseño estructurado](https://es.wikipedia.org/wiki/Dise%C3%B1o_estructurado) o en [diseño funcional](https://es.wikipedia.org/w/index.php?title=Dise%C3%B1o_funcional&action=edit&redlink=1) una función o un procedimiento, en [diseño orientado a objetos](https://es.wikipedia.org/wiki/Dise%C3%B1o_orientado_a_objetos) una clase. Esto sirve para asegurar que cada unidad funcione correctamente y eficientemente por separado. Además de verificar que el código hace lo que tiene que hacer, verificamos que sea correcto el nombre, los nombres y tipos de los parámetros, el tipo de lo que se devuelve, que si el estado inicial es válido entonces el estado final es válido

La idea es escribir casos de prueba para cada función no trivial o [método](https://es.wikipedia.org/wiki/M%C3%A9todo_(inform%C3%A1tica)) en el módulo, de forma que cada caso sea independiente del resto. Luego, con las [Pruebas de Integración](https://es.wikipedia.org/wiki/Pruebas_de_Integraci%C3%B3n), se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

**Características. -**

**Es automatizable:** donde no se requiere una intervención manual.

Cubren la mayor cantidad de código.

**Repetibles o reutilizables**: pruebas que se utilizan más de una vez.

Independientes: la ejecución de una prueba no afecta la ejecución de otra.

**Profesionales:** las pruebas deben ser consideradas igual que el código, con la misma profesionalidad, documentación, etc.

**Ventajas**

Las pruebas unitarias buscan aislar cada parte del programa y mostrar que las partes individuales son correctas, proporcionando cinco ventajas básicas:

* **Fomentan el cambio**, las pruebas unitarias facilitan la reestructuración del código (refactorización), puesto que permiten hacer pruebas sobre los cambios y verificar que las modificaciones no han introducido errores (regresión).
* **Simplifican la integración**, permiten llegar a la fase de integración asegurando que las partes individuales funcionan correctamente. De esta manera se facilitan las pruebas de integración.
* **Documentan el código**, las propias pruebas pueden considerarse documentación, ya que las mismas son una implementación de referencia de cómo utilizar el código.
* **Separación de la interfaz y la implementación**, la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro (ver pruebas mock).
* **Menos errores y más fáciles de localizar**, las pruebas unitarias reducen la cantidad de errores y el tiempo en localizarlos.
* **Pueden mejorar el diseño**, la utilización de prácticas de diseño y desarrollo dirigida por las pruebas (Test Driven Development o TDD) permite definir el comportamiento esperado en un paso previo a la codificación.
* **Puede ser la forma más simple de verificar el funcionamiento**, en situaciones como el desarrollo de una API o un componente que brinda servicios del cual no se cuenta aún con un cliente para consumirlos.

**¿Porque realizar pruebas unitarias?**

1.- Asegura calidad del código entregado.

2.- Ayuda a definir los requerimientos y responsabilidades de cada método en cada clase probada.

3.- Constituye una buena forma de ejecutar pruebas de concepto.

4.- Permite hacer refactoring tempranamente en el código.

5.- Permite encontrar errores o bugs tempranamente en el desarrollo.

**Limitaciones:**

Es importante darse cuenta de que las pruebas unitarias no descubrirán todos los errores del código. Por definición, sólo prueban las unidades por sí solas. Por lo tanto, no descubrirán errores de integración, problemas de rendimiento y otros problemas que afectan a todo el sistema en su conjunto. Además, puede no ser trivial anticipar todos los casos especiales de entradas que puede recibir en realidad la unidad de programa bajo estudio. Las pruebas unitarias sólo son efectivas si se usan en conjunto con otras pruebas de software.

**¿Cómo Trabaja?**

Está basado en aserciones, que son condiciones que debe cumplir el resultado dela ejecución del método para demostrar que su funcionamiento es correcto. Existen herramientas, frameworks, que facilitan este trabajo, una de ellas es el framework JUNIT.

**Ejemplo**

Si un método “Suma” De Una clase debe calcular la suma de dos números que se les pasa como parámetros, entonces la forma de probarlo es:

**resultado = objeto.suma(2,3);**

**objeto.assertEquals(5, resultado);**

Código que al ejecutarlo debe dar como resultado una ejecución Con luz “verde”,

Caso contrario se prende una luz “roja “mostrando los errores.

## 

## Cuando se crea las pruebas unitarias

Una prueba unitaria también se puede crear manualmente, sin utilizar la característica de generación. La mejor manera de hacerlo es agregar una prueba unitaria a un proyecto de prueba. De este modo, se crea un método de prueba que no contiene código, pero que tiene el atributo [TestMethod ()] aplicado. Para completar la prueba, se debe editar en su archivo de código fuente, que a su vez reside en un proyecto de prueba de la solución.

## Generar pruebas unitarias:

Las pruebas unitarias se pueden generar utilizando un cuadro de diálogo al que se tiene acceso a través de las ventanas de herramientas de Visual Studio. Cada prueba unitaria se crea como un método de C#, Visual Basic o Visual C++. Su código y sus atributos residen en un archivo de código fuente en un proyecto de prueba del mismo lenguaje. La nueva prueba se orienta al código específico que se desea probar. Puede generar una prueba unitaria para un método individual, varias pruebas para métodos que seleccione, o varias pruebas para todos los métodos de una clase o un espacio de nombres.

Puede generar pruebas unitarias desde el código fuente del proyecto actual. También puede generar pruebas unitarias desde un ensamblado del sistema de archivos, lo que resulta útil cuando el código fuente no está disponible.

El código de una prueba recién generada se compila pero, para que pueda proporcionar resultados útiles, primero se debe editar. Por ejemplo, podría editarlo para realizar asignaciones de variables o para personalizar instrucciones Assert. Las pruebas unitarias generadas contienen instrucciones TODO que indican qué líneas de código se deben editar.

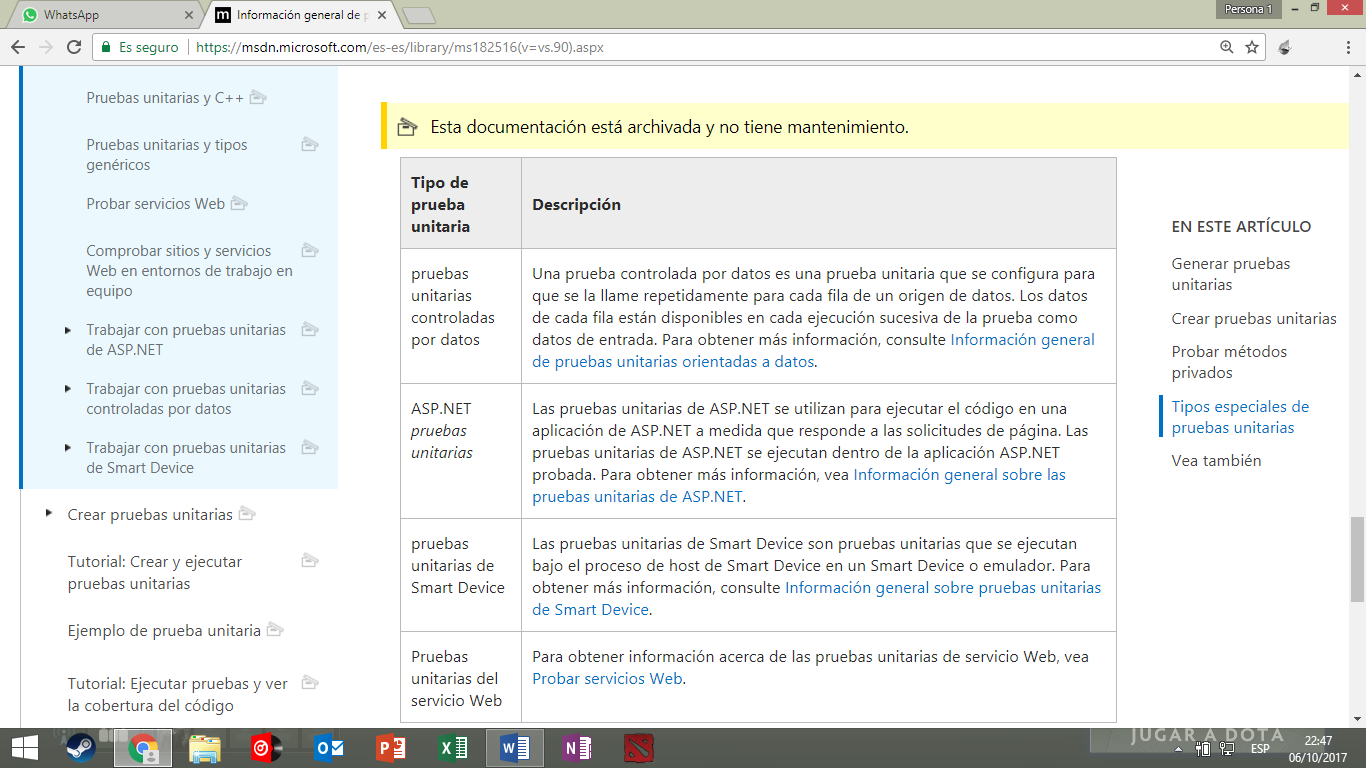
# 

# **Estructura de las pruebas unitarias**

Para que se reconozca la prueba unitaria en Herramientas para pruebas Team System, este archivo de código fuente debe residir en un proyecto de prueba, que a su vez sea parte de una solución de Visual Studio. Cuando se genera este proyecto o la solución completa, el proyecto de prueba se integra en un ensamblado que contiene la prueba unitaria ejecutable.

Todos los métodos de prueba unitaria están marcados con el atributo [TestMethod ()], <TestMethod ()> en Visual Basic, y son miembros de la clase [TestClass ()]. Cuando se genera una prueba unitaria, se observa que este espacio de nombres se incluye al principio del archivo generado, en una instrucción using o Imports.

**Tipos especiales de pruebas unitarias**



**Conclusiones**

* Las pruebas unitarias son parte de los entregables que toda aplicación debería considerar.
* No es necesario imprimir los resultados en consola basta con utilizar aserciones que se encargarán de comparar los resultados obtenidos frente a los esperados y si falla muestra automáticamente los resultados obtenidos que están fallando.
* La prueba unitaria permite un mejor trabajo en equipo y es considerado un método ágil.

**Referencias:**

http://es.wikipedia.org/wiki/Prueba\_unitaria

http://elisa.dyndns-web.com/~elisa/teaching/prog/oo/2011/codigo/junit\_demo.tar.gz

(Para el Prueba.java me base en el código ejemplo de la Dra. Schaeffer)

https://github.com/KentBeck/junit/downloads

http://maperz1130.files.wordpress.com/2010/09/tipos\_pruebas\_es.gif

https://msdn.microsoft.com/es-es/library/ms182516(v=vs.90).aspx

https://es.scribd.com/document/239246451/Pruebas-Unitarias