



**Área Académica de Ingeniería en Computadores**

**CE-4101 Especificación y Diseño de Software**

**Investigación 1**

**“Pruebas Unitarias”**

**Estudiantes:**

**Roy Acevedo Mendez**

**Rodolfo Bonilla Camacho**

**Profesor:**

**Daniel Madriz Huertas**

**Fecha de Entrega:**

**12/03/2021**

**I Semestre – 2021**

# Índice

<b>Índice</b>	<b>2</b>
<b>¿Qué son Pruebas Unitarias y sus beneficios y limitaciones?</b>	<b>3</b>
<b>Frameworks de pruebas unitarias para C#</b>	<b>4</b>
<b>Frameworks de pruebas unitarias para JavaScript o TypeScript</b>	<b>5</b>
<b>Ejemplo de prueba usando C#</b>	<b>6</b>
<b>Ejemplo de prueba usando JavaScript o TypeScript</b>	<b>8</b>
<b>Link a GitHub</b>	<b>9</b>
<b>Conclusiones</b>	<b>10</b>
<b>Bibliografía</b>	<b>11</b>

# ¿Qué son Pruebas Unitarias y sus beneficios y limitaciones?

Las pruebas unitarias son pequeñas verificaciones que se realizan para examinar si una sección de código funciona adecuadamente o en la manera en la que se espera que funcione.

En pocas consisten en separar un fragmento o parte del código y depurarlo hasta que su comportamiento sea el que se espera, dentro de lo posible. Este proceso se hace en repetidas ocasiones con diferentes partes del código hasta validar que cada pequeño fragmento haga su tarea de manera óptima, lo cual comúnmente convierte a que el código como un todo también debería funcionar de manera apropiada.

Técnicamente, las pruebas unitarias deberían llevarse a cabo durante la etapa de desarrollo del proyecto, debido a que de esta forma es más sencillo detectar un error y solucionarlo antes de que este se vuelva en un problema grande para el resto del trabajo. No obstante, en algunos casos se transfiere esta sección del proyecto a los encargados de QA (Quality Testing), quienes tienen como labor principal efectuar pruebas en el código o el producto antes de que este sea liberado en producción (YeePLY, 2019).

Según YeePLY en 2019, las pruebas unitarias son sumamente importantes en la metodología ágil. Estas aportan una gran cantidad de ventajas al utilizarse:

- Integran la lógica del código para todos los posibles casos (en teoría). Esto proporciona comprobar su correcto desempeño.
- Se utilizan como una pseudo-documentación del proyecto, debido a que permite comprobar de manera individual la comprensión del producto.
- Produce un código limpio y de alta calidad, con cada una de sus secciones definidas y bien estructuradas.
- Apoya a comprobar una parte del código dependientes de otra u otras de forma individual.
- Contribuyen a la detección de errores.

## Frameworks de pruebas unitarias para C#

Para C# hay varios frameworks que se pueden utilizar. Se pueden utilizar en proyectos compatibles con el suite .NET de Microsoft. En esta investigación se usará el framework MSTest V2.

MSTest V2 es una versión actualizada de su antecesor MSTest que fue un antiguo framework empleado para hacer algunas pruebas en proyectos de carácter .NET. La segunda versión es mejor en prácticamente en todos los aspectos y hoy en día es uno de los más utilizados por un conjunto de propósitos detalladas a continuación.

Según Roberts en 2018, en la actualidad este framework es muy usado en diferentes proyectos, puesto que al ser equipado por Microsoft y ser open-source, contribuye con la accesibilidad y manejabilidad. Unido a estas ventajas es cross-platform, lo que facilita su funcionalidad en variados sistemas operativos de forma orgánica.

Donde las funciones que MSTest V2 suministra se pueden apuntar son las siguientes (Roberts, 2018):

- Da funcionalidad típica de aserción, como la aserción de los valores de: String, Colecciones, Números , thrown-exceptions y entre otras..
- Otros framework usados actualmente, MSTest V2 proporciona la personalización del ciclo de vida de ejecución de la prueba, además de la ejecución del código de configuración adicional antes de que se ejecute cada prueba.
- Suministra la funcionalidad de producir que un solo método de prueba que se efectúa varias veces sea capaz de recibir diferentes variables de entrada secuencialmente.

# Frameworks de pruebas unitarias para JavaScript o TypeScript

Dentro de las herramientas que se pueden utilizar para realizar dichas pruebas, existen algunos frameworks que se especializan en ello, tales como Mocha/Chai, Jasmine y Jest. Para esta investigación y sus respectivas pruebas o ejemplos, se estará usando Mocha/Chai. Este framework mencionado anteriormente, nos permite realizar pruebas muy completas, demostrando una compatibilidad tanto para JavaScript como para TypeScript, además de poder trabajar en Node.js y en nuestro caso con Angular.

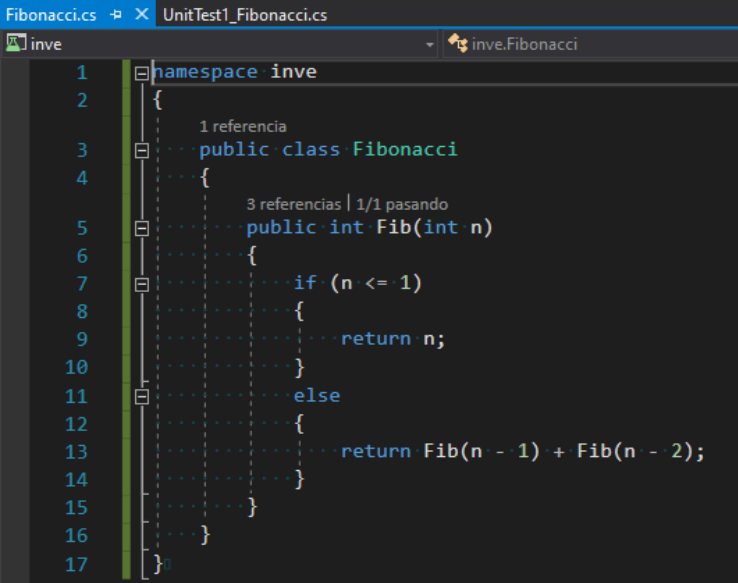
Al usar Mocha, nos brinda el uso de funciones reservadas como “describe” o “it” y que se debe usar en paralelo con las herramientas que nos provee Chai, como “expect”.

Los tipos de prueba que nos brinda esta herramienta son muy específicas, ya que le da un enfoque controlado a ciertas partes de un código a validar, en especial dándole mucho énfasis al orden o el proceso que debe tener dicho código. En otras palabras, Mocha provee funciones que son ejecutadas de acuerdo a un orden específico.

Algunas funcionalidades y herramientas:

- Funciones basadas en un orden funcional
- Banco de pruebas (Test Suites), esto para testear individualmente casos relacionados a una misma funcionalidad.
- Herramientas de limpieza de código, con esto es posible crear un entorno aislado de prueba que no involucre código externo.
- Uso de Assertions con el fin de especificar posibles valores a ser retornados, lo que se espera que retorne o haga ciertas pruebas.
- **describe()**: Información de la prueba a realizar.
- **it()**: Descripción de lo que debería hacer dicha prueba.
- **expect()**: Lo que se espera que haga.
- **before()**: Lo que se define aquí es ejecutado en primer orden
- **after()**: Aquí se especifica lo que se ejecutará de al final o después de todos los casos.

## Ejemplo de prueba usando C#

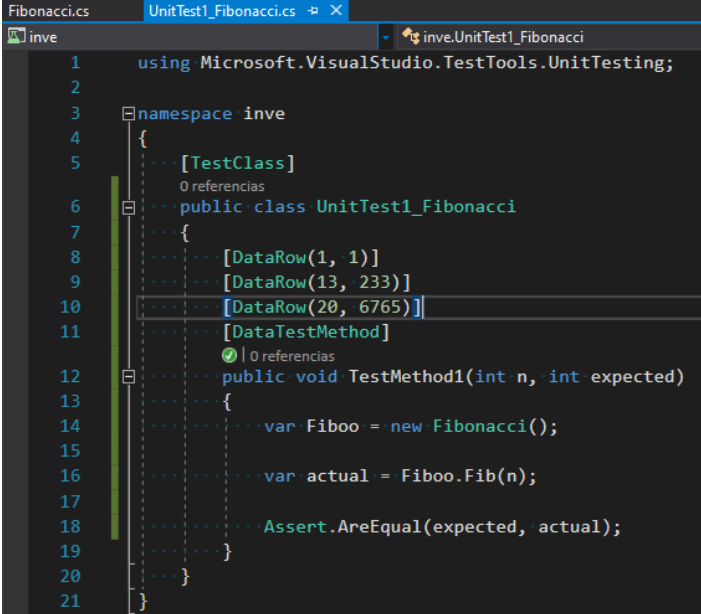


```
1 namespace inve
2 {
3     1 referencia
4     public class Fibonacci
5     {
6         3 referencias | 1/1 pasando
7         public int Fib(int n)
8         {
9             if (n <= 1)
10             {
11                 return n;
12             }
13             else
14             {
15                 return Fib(n - 1) + Fib(n - 2);
16             }
17         }
18     }
19 }
```

Figura 1. Método para determinar Fibonacci de un número, implementado en C#.

El método de la Figura 1 se encarga de encontrar el Fibonacci de un número dado. Su condición de parada es cuando  $n$  sea menor o igual a 1. Es un método recursivo y se cada sucesión se calcula como  $Fib(n - 1) + Fib(n - 2)$ .

A continuación, se muestran las pruebas unitarias realizadas para este método:

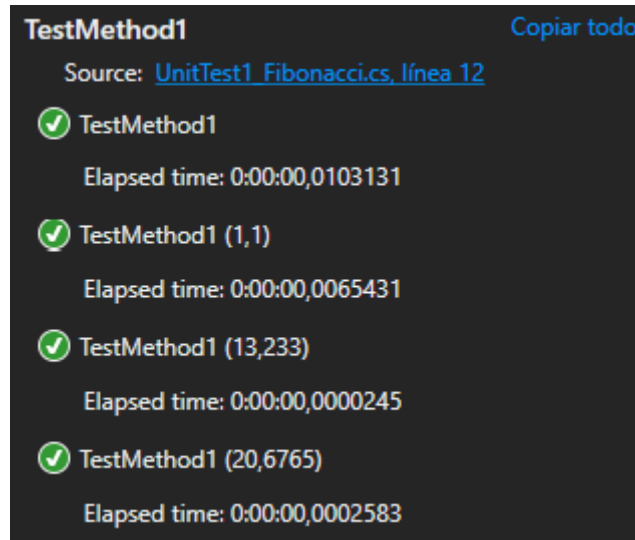


```
1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2
3 namespace inve
4 {
5     [TestClass]
6     public class UnitTest1_Fibonacci
7     {
8         [DataRow(1, 1)]
9         [DataRow(13, 233)]
10        [DataRow(20, 6765)]
11        [DataTestMethod]
12        public void TestMethod1(int n, int expected)
13        {
14            var Fiboo = new Fibonacci();
15            var actual = Fiboo.Fib(n);
16            Assert.AreEqual(expected, actual);
17        }
18    }
19 }
```

Figura 2. Prueba unitaria del método de Fibonacci recursivo, implementada en MSTest V2.

Como se puede ver en la Figura 2, las pruebas unitarias consisten en introducir un número al método y un de número esperado (con el resultado de la sucesión de Fibonacci).

Visto que es método matemático probar un número de repeticiones ascendente es permite verificar que tan eficaz el método (hace lo que debería) y que tan eficiente (desempeña su función en un tiempo moderado).

The image shows a screenshot of the MSTest V2 test runner interface. At the top, it says 'TestMethod1' and 'Copiar todo'. Below that, it shows the source as 'UnitTest1 Fibonacci.cs, línea 12'. There are four test results listed, each with a green checkmark icon. The first is 'TestMethod1' with an elapsed time of 0:00:00,0103131. The second is 'TestMethod1 (1,1)' with an elapsed time of 0:00:00,0065431. The third is 'TestMethod1 (13,233)' with an elapsed time of 0:00:00,0000245. The fourth is 'TestMethod1 (20,6765)' with an elapsed time of 0:00:00,0002583.

Test Name	Elapsed time
TestMethod1	0:00:00,0103131
TestMethod1 (1,1)	0:00:00,0065431
TestMethod1 (13,233)	0:00:00,0000245
TestMethod1 (20,6765)	0:00:00,0002583

Figura 3. Resultado obtenido de la prueba unitaria del método de Fibonacci recursivo, implementada en MSTest V2.

En la la Figura 3 se muestra los resultados obtenidos para cada de las pruebas unitarias hechas para el método de fibonacci donde se muestra que es correcto el resultado real con el esperado. Y también se nota el tiempo que tardó en ser ejecutado.

## Ejemplo de prueba usando JavaScript o TypeScript

```
src > TS usercontroller.ts > userMNG > constructor
1  //---Imports---//
2  import { userData } from './data';
3  //-----Clase principal-----//
4  export class userMNG{
5
6      username: any;
7      password: any;
8      data = new userData();
9      usersList = [];
10
11     //Constructor//
12     constructor(){
13         this.username = "";
14         this.password = "";
15     }
16
17     //Funcion para agregar usuario a una lista//
18     addUser(name: any, pass: any){
19
20         this.username = name;
21         this.password = pass;
22
23         this.data.informacion.username = this.username;
24         this.data.informacion.password = this.password;
25
26         this.usersList.push(this.data);
27     }
28 }
```

Figura 4. Código de ejemplo desarrollado en TypeScript

En la figura 4, se logra apreciar el código de prueba, el cual es un simple manejador de usuarios. Dicha clase contiene variables como el nombre de usuario, la contraseña y una lista. La funcionalidad es simple, crear y guardar el usuario en una lista.

Para este código, verificaremos las pautas fundamentales y básicas, de Mocha y Chai, como lo son el uso de “it”, “describe” y “expect”.

```
test > TS usercontroller.ts > ...
1  //-----Imports-----//
2  import { userMNG } from '../src/usercontroller';
3  import { expect } from 'chai';
4
5  describe('userManagement', () => {
6      //Primer Test//
7      it('Se debe iniciar las variable con "", indicando que no contienen valores previos', () => {
8          let userMng = new userMNG();
9          expect(userMng.username).to.equal("");
10         expect(userMng.password).to.equal("");
11     })
12     //Segundo Test//
13     it('Deberia de verificar que el objeto recién creado, haya sido agregado exitosamente a la lista de usuarios', () => {
14         let userMng = new userMNG();
15         userMng.addUser("roy", "abc123")
16         expect(userMng.usersList.find(e => e.informacion.username === "roy")).to.equal(userMng.data);
17     })
18 })
19 })
```

Figura 5. Desarrollo de la prueba.



Para esta prueba presente en la figura 5, se probarán dos cosas, una, que al crearse dicha clase, no haya valores almacenados en las variables y dos, que dicho usuario sea agregado correctamente a la lista y exista en la lista.

Se describe la información de la prueba con el uso de **describe()**, posterior a eso, hacemos uso de **it()**, el primero debería de verificar que los valores de usuario y contraseña, estén vacíos, ahí el uso de **expect(userMng.username).to.equal("")** y en el segundo **it()**, se validará que dicho usuario creado, se haya agregado correctamente a la lista, realizando una búsqueda y comparándolo con el valor en tiempo real en la clase definido como **data**.

```
C:\Users\roygt\UnitTest>npm test
> unit-test@0.0.0 test C:\Users\roygt\UnitTest
> mocha -r ts-node/register test/*.ts

userManagement
  ✓ Se debe iniciar las variable con "", indicando que no contienen valores previos
  ✓ Deberia de verificar que el objeto recién creado, haya sido agregado exitosamente a la lista de usuarios

2 passing (49ms)

C:\Users\roygt\UnitTest>
```

Figura 6.Resultados de la pruebas.

Se logra apreciar en la figura 6, que las dos pruebas realizadas, pasaron con éxito, validando la inicialización con valores vacíos y la adición a la lista de usuarios.

**Link a GitHub**

## Conclusiones

Se concluye que las pruebas unitarias son una herramienta que utilizadas de la forma adecuada puede ahorrar muchos inconvenientes en los sistemas y métodos de desarrollo de un proyecto. Asimismo no solamente impide errores, por tanto ayudan a crear código más eficiente y limpio.

Cabe resaltar que son muchos los frameworks y bibliotecas especializadas a los cuales se tiene disponibilidad como estudiantes, dado que muchos de ellos, incluso los utilizados por grandes empresas, y son de código abierto (open source). Esto representa que incluso usuarios que practican la programación por pasatiempos o que no tengan accesibilidad a ciertas licencias pagadas puedan utilizarlas.

Adicionalmente, se destaca que el procedimiento de implementar pruebas unitarias es un compromiso por parte de un equipo de desarrolladores o un trabajador individual. Los instrumentos que existen son muy buenos, no obstante, están sujetos a cada programador asumir el estándar de calidad en su trabajo para garantizar dichas pruebas.

Se logra comprobar la importancia de las pruebas unitarias a la hora de validar o probar segmentos de código en un proyecto, guiando su desarrollo por el buen camino. Se deben realizar desde el nacimiento del proyecto hasta la llegada de su conclusión, eso garantiza un crecimiento funcional al tener una evaluación concisa de lo que se espera que haga nuestro código y no tener que arrastrar errores desde sus inicios, que puedan generar inconvenientes en el futuro.

# Bibliografía

YeePLY. (2020) ¿Qué son las pruebas unitarias y cómo llevar una a cabo? YeePLY.

Obtenido de: <https://www.yeeply.com/blog/que-son-pruebas-unitarias/>

Barré, G. (5 de febrero, 2018) *MSTest v2: Data tests Meziantu's Blog*.

Obtenido de: <https://www.meziantou.net/mstest-v2-data-tests.htm>.

Roberts, J. (2018) MSTest V2. Don't Code Tired.

Obtenido de: <http://dontcodetired.com/blog/post/MSTest-V2>

Microsoft. (2020) Microsoft Test Framework "MSTest V2" GitHub.

Obtenido de: <https://github.com/Microsoft/testfx>

Helper, D. (16 de setiembre, 2020) *MSTest V2: First Impressions. DZone*.

Obtenido de: <https://dzone.com/articles/mstest-v2-first-impressions>

CodeCademy. (n.d.). *Introduction to Testing with Mocha and Chai*. codecademy. Retrieved

Marzo 11, 2021, from <https://www.codecademy.com/articles/bapi-testing-intro>