# **¿Qué son las pruebas unitarias y cómo llevarlas a cabo?**

Las pruebas unitarias consisten en **aislar una parte del código y comprobar que funciona a la perfección**. Son pequeños *tests* que validan el comportamiento de un objeto y la lógica.

El *unit testing* suele realizarse durante la fase de desarrollo de aplicaciones de software o móviles. Normalmente las llevan a cabo los **desarrolladores**, aunque en la práctica, también pueden realizarlas los **responsables de QA**.

Hay una especie de mito respecto a las pruebas unitarias. Algunos desarrolladores están convencidos de que son una pérdida de tiempo y las evitan buscando ahorrar tiempo.

**Nada más alejado de la realidad.**

Con ellas se **detectan antes errores** que, sin las pruebas unitarias, no se podrían detectar hasta fases más avanzadas como las pruebas de sistema, de integración e incluso en la beta.

Realizar pruebas unitarias con regularidad supone, al final, un ahorro de tiempo y dinero.

## **Motivos para realizar un test unitario**

Algunos motivos:

* Las pruebas unitarias demuestran que la **lógica del código** está en buen estado y que funcionará en todos los casos.
* Aumentan la **legibilidad del código** y ayudan a los desarrolladores a entender el código base, lo que facilita hacer cambios más rápidamente.
* Los test unitarios bien realizados sirven como **documentación** del proyecto.
* Se realizan en **pocos milisegundos**, por lo que podrás realizar cientos de ellas en muy poco tiempo.
* Las *unit testing* permiten al desarrollador **refactorizar el código** más adelante y tener la garantía de que el módulo sigue funcionando correctamente. Para ello se escriben casos de prueba para todas las funciones y métodos, para que cada vez que un cambio provoque un error, sea posible identificarlo y repararlo rápidamente.
* La **calidad final del código** mejorará ya que, al estar realizando pruebas de manera continua, al finalizar el código será limpio y de calidad.
* Como las pruebas unitarias dividen el código en pequeños fragmentos, es posible **probar distintas partes del proyecto** sin tener que esperar a que otras estén completadas.

## **Las 3 A’s del unit testing**

Para llevar a cabo buenas pruebas unitarias, deben estar estructuradas siguiendo las tres A del Unit Testing. Se trata de un **concepto fundamental respecto a este tipo de pruebas**, que describe un proceso compuesto de tres pasos.

* Arrange (organizar). Es el primer paso de las pruebas unitarias. En esta parte se **definen los requisitos** que debe cumplir el código.
* Act (actuar). Es el paso intermedio de las pruebas, el momento de **ejecutar** el *test* que dará lugar a los resultados que deberás analizar.
* Assert (afirmar). En el último paso, es el momento de comprobar si los resultados obtenidos son los que se esperaban. **Si es así, se valida y se sigue adelante**. Si no, se corrige el error hasta que desaparezca.

### **Cómo llevarlas a cabo**

El proceso de los tests unitarios puede realizarse de manera manual, aunque lo más común es **automatizar el procedimiento a través de herramientas**.

Hay muchas opciones disponibles, que varían en función del lenguaje de programación que se esté utilizando. Estos son algunos ejemplos de este tipo de herramientas que te ayudarán con las pruebas.

* [xUnit](https://xunit.net/): se trata de una herramienta de pruebas unitarias para el framework **.NET**.
* [~~Junit~~](https://junit.org/junit5/)~~: es un conjunto de bibliotecas para realizar pruebas unitarias de aplicaciones~~ **~~Java~~**~~.~~
* [NUnit](https://nunit.org/): inicialmente portado desde JUnit, NUnit 3 se ha reescrito por completo para dotarlo de nuevas características y soporte para una amplia gama de plataformas **.NET**.
* [~~PHPUnit~~](https://phpunit.de/)~~: entorno de pruebas unitarias en el lenguaje de programación~~ **~~PHP~~**~~.~~

Al utilizar estas herramientas, **se codifican los criterios en la prueba** que verificarán si el código es o no correcto. Durante la fase de ejecución, la herramienta puede detectar las pruebas con errores.

Si alguno de estos errores es grave, puede **detener pruebas posteriores** que iban a realizarse a continuación.

## **TDD y pruebas unitarias**

El ***test driven development* (TDD) o desarrollo guiado por pruebas** implica desarrollar las pruebas unitarias a las que se va a someter el software antes de escribirlo.

De esta manera, el desarrollo se realiza atendiendo a los requisitos que se han establecido en la prueba que deberá pasar. El objetivo es conseguir un **código limpio** que funcione.

Con esta metodología, se escoge un requisito de la lista y se plantea una prueba que se ejecuta para comprobar que falla. Si no falla puede ser porque no se ha planteado correctamente o porque la función ya estaba implementada.

A continuación, se escribe el código que haga posible pasar la prueba de la manera más simple posible, se ejecutan las pruebas y, si todo es correcto, **se refactoriza el código para eliminar las partes duplicadas**.

Así se puede tachar ese requisito de la lista y seguir avanzando con el desarrollo.

**Hay 3 tipos de Unit Testing**

* NUTest
* MSTest
* XTest

**EJEMPLO PRÁCTICO:**

Ejemplo sobre MStest. **EN REPOSITORIO**

https://github.com/gitgamba/.Net-Grupo8/tree/master/TPS/UnitTesting

--------------------------------------------------------------------------------

Rafa 19/05/21:

**xUnit.Net**

Basado en .Net Framework. Creada por los mismos que NUnit , donde el objetivo era crear un mejor framework que AGREGAR features(características) al NUnit framework.

xUnit esta enfocada en la comunidad.

Razones por las cuales decidieron arrancar desde cero:

* Tags no son mas usados.
* Attributes como ~~[setup][tearDown]~~ No se utilizan mas, para darle mas optimización al codigo.
* Para inicialización, constructor of test class es usado, Utilizan interfaces. esto también cubre mucho escribir cleaner tests. Esto hace que C# unit testing framework sea una mejor opción cuando se tiene **SELENIUM AUTOMATION TESTING** asi como mas robusto y extensible (como mas lleno y amplio quiere decir)
* La gran diferencia entre NUnit, XUnit y MSTest , es que XUnit es mas extensible (o mas amplio)
* “The [Fact] attribute is used instead of [Test] attribute. Non-parameterized tests are implemented under the [Fact] attribute, whereas the [Theory] attribute is used if you plan to use parameterized tests.”
* In NUnit and MSTest, la clase que contiene el test esta debajo de [testclass] attribute. Como no era muy robusto
* Hay una lista de attributes/annotations mas usadas.

**MSTest:**

Viene por defecto que esta integrada con Visual Studio. La version inicial de MSTest (V1) no era una opensource pero la V2 de MSTest si, esta era hosteada en GitHub(la cuál esta mas para repositorios) . También podía ser usada para “data driven testing”.

MSTest V2 tiene multiplataforma y ademas puede ser extendida usando custom test attributes y custom asserts.

Paso de ser un “in box” testing framework a ser un framework que gano mas amplitud en adopcion por los desarrolladores de la comunidad,

-----------------------------------------------------------------------------------------

XUnit mejor para los test ya que hace que no haya tanta dependencia por su reutilización ya que inicializa, ejecuta y descarta

MSTest orientado a los programadores junior (corre con la ventaja de que esta integrado en visual studio)

XUnit te desprende de tener que hacer todo estructurado

::::::::::::::::::::::::::::::

para MSTEST ver link de esta pagina (y con esto me apoyo para el ejemplo) <https://www.genbeta.com/desarrollo/ms-test-el-framework-de-test-de-visual-studio-2010>

<https://www.lambdatest.com/blog/most-complete-mstest-framework-tutorial-using-net-core-2/>