Frameworks de pruebas unitarias

Fabrizio Dumani (2015182277),
Bryan Masís (2013031110),
Isaac Núñez (2013030911)

Viernes 23 de marzo de 2018

Instituto Tecnológico de Costa Rica

Área académica de ingeniería en computadores

Especificación y diseño de software, CE4101

-

Índice

1. Introducción

Al trabajar en un programa es necesario tener la seguridad de que este va a funcionar correctamente cada vez que se quiera utilizar. El problema se viene a dar cuando es necesario agregar al código, y en especial cuando es un proyecto grande al que aportan múltiples programadores, se ve la posibilidad de que un nuevo agregado afecte negativamente otras partes del sistema accidentalmente. Para evitar este tipo de inconvenientes se utilizan pruebas unitarias, las cuales tienen el objetivo de verificar que todo esté funcionando correctamente de forma automática.

En la siguiente investigación se estudiará la utilidad de las pruebas unitarias, además de realizar casos de ejemplos que permitan comprender su funcionamiento. Se programará en los lenguajes C# y Javascript para los cuales se utilizarán los marcos de xUnit y Mocha and Chai respectivamente.

2. Pruebas unitarias

2.1. ¿Qué es una prueba unitaria?

El uso de pruebas unitarias es un método utilizado para probar requerimientos aislados en el código de un programa. Es necesario que como desarrollador se sepa exactamente que se debe de esperar de diferentes secciones del sistema y así acreditar que su funcionamiento sea consistente. Se espera que tenga las siguientes características:

- Realiza pruebas sobre pequeños trozos de código.
- Es completamente aislado de otras secciones del sistema, por lo que funciona exclusivamente para la sección deseada.
- Es un proceso automatizado que permite controlar cuando se quieren ejecutar las pruebas sobre los resultados.
- Es consistente, independiente de cuantas veces se repita la prueba el resultado tiene que ser el mismo.

- Debe ser rápido de desarrollar, es imperativo que el programador pueda enfocar su tiempo al funcionamiento del sistema y no invertir demasiados recursos en crear las pruebas para este. [2]

2.2. xUnit

Framework para pruebas unitarias en .net C#. Bastante extensivo y fácil de utilizar, de sintáxis sencilla y fácil de entender. Tiene varios tipos de pruebas las cuales puede identificar mediante atributos. Tiene varios sin embargo se puede centrar la atención en:

- Fact: Para métodos que deberían ser siempre ciertos y no tienen argumentos.
- Theory: para métodos que son ciertos sólo con ciertos datos y tienen argumentos.

Para poder utilizarlo es importante instalar el paquete desde el NuGet (se debe instalar en cada proyecto aún si están en la misma computadora).xUnit tiene dos métodos de ejecución, los cuales difieren en que el primero utiliza la consola para ejecutar y mostrar los resultados de las pruebas unitarias y el segundo(sólo en visual studio) utiliza el visor de pruebas para la solución y los muestra de forma más agradable al usuario.

Ambos son paquetes NuGet que deben ser instalados en la solución además del xUnit en sí. La ejecución en consola toma mucho tiempo ya que requiere abrir una terminal CMD o powershell, navegar a la ruta donde está guardado el proyecto, una vez dentro navegar donde está el dll del xUnit y correrlo desde ahí mientras que la ejecución desde el visor de pruebas en Visual Studio es muchísimo más rápida, ya que el programa busca las pruebas en el proyecto, las enlista y permite correrlas todas de una vez sin más. [1]

2.3. Mocha y Chai

Este framework funciona sobre *node.js* y provee soporte para la versión en el buscador. Vuelve las pruebas unitarias asincrónicas fáciles de implementar gracias a la cantidad de caracterísiticas presentes. Asímismo, permite la integración con varias *assertion libraries*, en este caso Chai.[3]

Por otro lado, Chai es una *Behavior Driven Development and Test Driven Development assertion library* diseñada para funcionar bajo node o el buscador y de fácil integración con los frameworks de pruebas. Las caracterísiticas del API dependen de la aplicación a la que se quiera enfocar. Tanto así que el *assertion style* presenta dos perspectivas de acuerdo a los tipos de desarrollo usado: para BDD se tiene *expect/should* y para TDD se tiene *assert.*[4]

Cabe aclarar que Mocha no posee *assert* nativo, sino que utiliza el método heredado por *Node.js* o sino permite la integración con varias bibliotecas que brinden dicha funcionalidad.

Las diferentes funcionalidades que presenta Mocha para el desarrollo de *unity test* son:

- Permite las pruebas dentro de código asincrónico y sincrónico. En las pruebas asincrónicas permite la intgegración con *promise* - que se refiere al retorno de un bloque asincrónico y su valor[5].
- Además permite el uso de *hooks*, estos establecen condiciones antes y después que una función se ejecute, con el fin de prepararlsa y hacer limpieza al finalizar las pruebas.
- Permite verificar solamente una parte del test mediante la función *.only()*. Si dicha función se especifica en un describe anidado, igualmente todos los tests se ejecutarán, si solo se desea un test, este se debe especificar mediante *it.only()*.
- Por otra parte, si se desea ejecutar todos los test menos uno, al que se desea excluir, se le aplica la función *describe.skip()*. Si se desea excluir un caso de prueba en específico, se debe usar *it.skip()*. Es importante denotar que la mejor práctica es usar *skip()* en vez de comentar el código.
- Posee varias interfaces disponibles de acuerdo al *assertion style*. Para BDD presenta *describe(), context(), it(), specify(), before(), after(), beforeEach()* y *afterEach()*. Donde *context()* es sólo un sinónimo para *describe()* y así mismo *specify()* lo es para *it()*. En el caso de TDD provee *suite(), test(), suiteSetup(), suiteTeardown(), setup()* y *teardown()*

Para Chai en el ámbito de *expect/should*, provee la herramienta de *getters* enlazados, esto permite que la validación sea más legible, lo único en lo que difieren es en la construcción

del *assert*. Para *expect*, la forma es: `expect(<function>).<cadena de getters>`; en cambio para *should* se aplica la foma de: `<function>.should.<cadena de getters>`. La cadena de getters disponible es: *to, be, been, is, that which, and, has, have, with, at, of, same, but, does* y *not* niega los getters siguientes.

Para el caso de *assert*, se tiene tiene la restricción que no puede ser entrelazado, esto significa que solo puede ir acompañado de un método. Posee numerosas funciones, por mencionar algunas: *equal(), strictEqual(), isTrue()* (ver más en [4]).

Nota: Es importante hacer notar que estas pruebas fueron realizadas en Linux (Linux Mint 18.3 LTS).

3. Conclusión

La pruebas unitariar muestran ser una herramienta necesaria para el manejo de código ya que permiten generar systemas que funcionan consistentemente, a pesar de ser transformados conforme el proyecto avanza. Es importante considerar que este método se tiene que utilizar en secciones pequeñas de código y debe de ser rápido de implementar, así permite al desarrollador concentrarse en generar un producto funcional con mínimo esfuerzo en producir la pruebas necesarias.

xUnit como framework para pruebas en c# es bastante completo y simple de utilizar, además de que su implementación con visual studio hace que ejecutar todas las pruebas sea rápido y permite al usuario consultar detalles del error.

El uso de Mocha + Chai provee una suite poderosa para realizar *unity test*, debido a su gran variedad de posibilidades en cuanto pruebas y validaciones gracias a la facilidad de integración entre ambos en un *test framework*.

Referencias

[1] xUnit.com, *Getting started with xUnit.net (.NET Core / ASP.NET Core)*. Available: `http://xunit.github.io/docs/getting-started-dotnet-core`. [Accessed: 21-3-2018].

[2] D. Rojas, *Qué es un unit test*, 2012. Available: `https://msdn.microsoft.com/es-es/communitydocs/alm/unit-test`. [Accessed: 21-3-2018].

[3] (s.f) *Mocha - the fun, simple, flexible JavaScript test framework* Available at: `https://mochajs.org/` [Accessed: 19-3-2018]

[4] (s.f) *Chai Assertion Library* Available at: `http://www.chaijs.com/` [Accessed: 19-3-2018]

[5] (s.f) *Promise - Javascript — MDN* Available at: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise` [Accessed: 19-3-2018]