Desarrollo de Web Api’s con .NET

Pruebas automatizadas

Un poco de repaso...

- ¿Para qué sirven los patrones de arquitectura? ¿Cuáles recuerdas?
- ¿Cómo organiza las clases MVC?
- ¿Qué es una API?
- ¿Qué relación/diferencia encuentran con REST?

¿Qué vamos a ver hoy?

- Qué es una prueba automatizada
- Qué diferentes tipos de pruebas automatizadas hay
- Qué valor nos aportan las pruebas automatizadas

¿Qué es una prueba automatizada?

Pruebas automatizadas

Consiste en el uso de software especial (separado del software productivo) para controlar la ejecución de pruebas y la comparación entre los resultados obtenidos y los resultados esperados.

Permiten incluir pruebas repetitivas y necesarias dentro de un proceso formal de pruebas.

Al automatizar pruebas de software se persigue el objetivo de simplificar el trabajo, repetitivo o complejo, haciéndolo efectivo y más productivo.

Manual vs automático

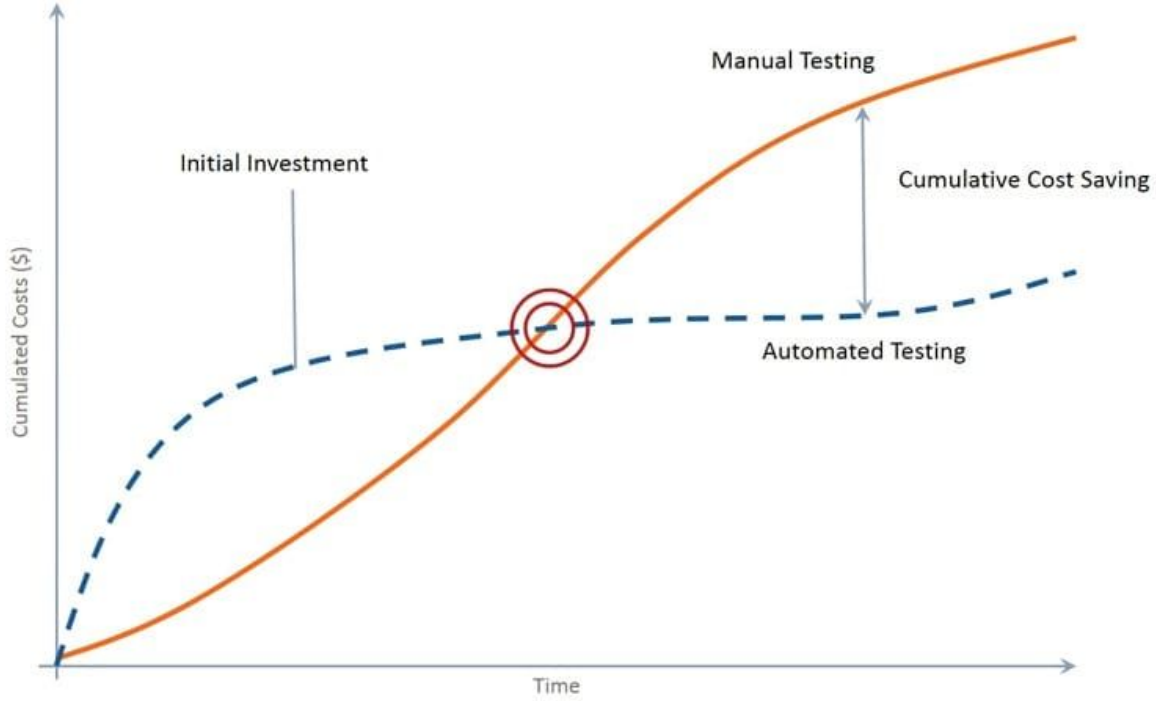
Aunque existen diferencias entre pruebas automatizadas y pruebas manuales, no son mutuamente excluyentes. Más bien se consideran tareas complementarias en la búsqueda de una mejor calidad de software.

Si pensamos en el retorno de la inversión de las pruebas, probar una nueva funcionalidad manualmente le permite conocer más sobre la aplicación, a bajo costo y rápidamente.

A medida que se adquiere conocimiento, el inventario de pruebas aumenta. En consecuencia, el costo también sube para las pruebas manuales.

Por otro lado, la automatización tiene un costo inicial más alto que disminuye a medida que avanza. Este comportamiento se puede ver a continuación

Manual vs automático



Elementos

SUT (Subject under Test): bloque de código que implementa la característica que estemos probando. El SUT se define siempre desde la perspectiva de la prueba.

Test Case: procedimiento para validar/verificar el **SUT**.

DOC (Depended-on-Component): partes de la aplicación que no estamos verificando en una prueba en particular de las que depende el **SUT**.

Pruebas unitarias

Una prueba unitaria es una prueba que ejercita componentes o métodos de software individuales, también conocidos como "unidad de trabajo".

Estas solo deberían probar código que pueda controlar el desarrollador. No se usan para comprobar problemas con la infraestructura.

Estos problemas incluyen la interacción con bases de datos, sistemas de archivos y recursos de red.

Pruebas de integración

Una prueba de integración se diferencia de una prueba unitaria en que ejercita la capacidad de dos o más componentes de software de funcionar juntos, a lo que se conoce también como su "integración".

Estas pruebas operan en un espectro más amplio del sistema sometido a prueba, mientras que las pruebas unitarias se centran en componentes individuales.

Las pruebas de integración suelen incluir problemas con la infraestructura.

Pruebas de carga

El objetivo de una prueba de carga es determinar si un sistema puede controlar una carga especificada.

Por ejemplo, el número de usuarios simultáneos que usan una aplicación y la capacidad de esta de controlar las interacciones de forma más responsable.

Herramientas de prueba

- Visual Studio
- xUnit
- NUnit
- MSTest

Documentación oficial

¿Porqué hacer pruebas unitarias?

La escritura de pruebas unitarias reporta muchos beneficios: ayudan con la regresión, proporcionan documentación y facilitan un buen diseño.

Pero también son difíciles de leer y, si son frágiles, pueden causar estragos en el código base.

Protección frente a regresión

Los defectos de regresión son aquellos que se presentan cuando se realiza un cambio en la aplicación.

Con las pruebas unitarias, es posible volver a ejecutar el conjunto completo de pruebas después de cada compilación o incluso después de cambiar una línea de código.

Eso da confianza en que el nuevo código no interrumpa la funcionalidad existente.

Documentación ejecutable

Si se tiene un conjunto de pruebas unitarias con un nombre adecuado, cada prueba debe ser capaz de explicar con claridad el resultado esperado de una acción determinada.

Además, debe ser capaz de comprobar que funciona.

Calidad del código

Cuando el código está estrechamente acoplado, puede resultar difícil realizar pruebas unitarias. Sin crear pruebas unitarias para el código que se está escribiendo, el acoplamiento puede ser menos evidente.

Al escribir pruebas para el código, este se desacopla naturalmente, ya que, de otra forma, sería más difícil de probar.

Características de una buena prueba unitaria

- Rápida. Las pruebas unitarias deberían tardar muy poco tiempo en ejecutarse. Milisegundos.
- Aislada. Las pruebas unitarias son independientes, se pueden ejecutar de forma aislada y no tienen ninguna dependencia en ningún factor externo (por ej, bases de datos).
- Reiterativa. La ejecución de una prueba unitaria debe ser coherente con sus resultados, es decir, devolver siempre el mismo resultado si no cambia nada entre ejecuciones.
- Autocomprobada. La prueba debe ser capaz de detectar automáticamente si el resultado ha sido correcto o incorrecto sin necesidad de intervención humana.
- Oportuna. Una prueba unitaria no debe tardar un tiempo desproporcionado en escribirse en comparación con el código que se va a probar.

Cobertura de código

Un alto porcentaje de cobertura de código suele ir asociado a una mayor calidad del código.

Pero la medida en sí no puede determinar la calidad del código.

La configuración de un objetivo de porcentaje de cobertura de código excesivamente ambicioso puede ser contraproducente

Resultado de pruebas

- Verde, ¡pasa!
- Falso negativo
- Rojo, ¡no pases!
- Falso positivo

Verde, ¡pasa!

Exito de Prueba donde los resultado esperado son los resultado arrojados por el **SUT** en la ejecución del caso de prueba.

Falso negativo

Escenario donde se produce un éxito en la prueba incluso sin que el SUT esté funcionando apropiadamente, o sea con fallos y/o errores que se escapan.

La recomendación es localizar el error extendiendo la cobertura de la red de pruebas con aquellas que detectan el error que se escapó.

Rojo, ¡no pases!

Fallo en la Prueba donde los resultados esperados no son los resultados arrojados en la ejecución del **SUT** en la ejecución del caso de prueba.

Error en la Prueba donde se produce un error (excepción) en la ejecución de la prueba que se ejecuta sobre el SUT. Por tanto, puede ser producido por ambas partes.

Fáciles de detectar porque las causas de los problemas suelen ser muy locales.

Falso positivo

Peligro donde se produce un fallo o error incluso con el SUT funcionando apropiadamente.

O sea, fallos o errores en las pruebas.

Habrá que arreglar las pruebas, quizás ajustando el SUT y/o las aserciones.

Y… ¿Si encontramos un error en el código?

¡No continúes con el desarrollo! No modifiques ni añadas comportamiento al SUT.

Es muy importante mantener el software inestable el menor tiempo posible o en un breve espacio de tiempo.

Equilibrio en las pruebas

Es crucial para mantener una alta productividad de pruebas, lograr los máximos resultados con el mínimo esfuerzo.

Un buen caso de prueba es cuando es probable que detecte un error que no se ha visto todavía.

Debe tratarse de ejecutar las pruebas correctas para encontrar los errores más importantes antes de que los errores de menor importancia.

Debe tratarse de descubrir los errores tan tempranamente como sea posible.

¡¡A trabajar!!