NTT DATA

Curso Vacacional Automatización de Pruebas

22 Diciembre 2022 – Clase 3

FUTURE AT HEART

Contenido

¿Que esperan de este curso?

Modulo I – Introducción la calidad de software

- Calidad de software
- Tipos de pruebas
- Automatización de pruebas

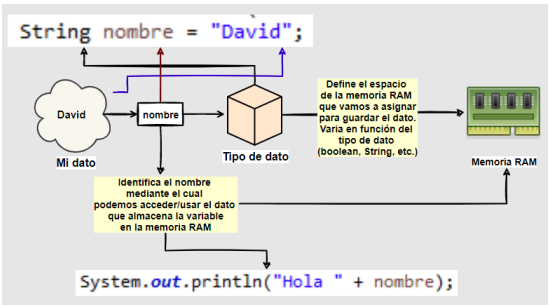
Modulo II – Introducción a Java

- Que es Java
- Conceptos de programación
- Instalación Ambiente (IDE, MVN)
- Creación de primer proyecto
- Descarga de dependencias (Selenium , Junit)
- Ejecutar primer script

Modulo III – Automatización en Java con Selenium

- Componentes del FW
- BDD (Gherkin, Steps, Métodos)
- Identificadores o Selectores
- Inspeccionar objetos
- Tipos de Objetos
- Xpath (Tipos)
- Introducción a testing Continuo

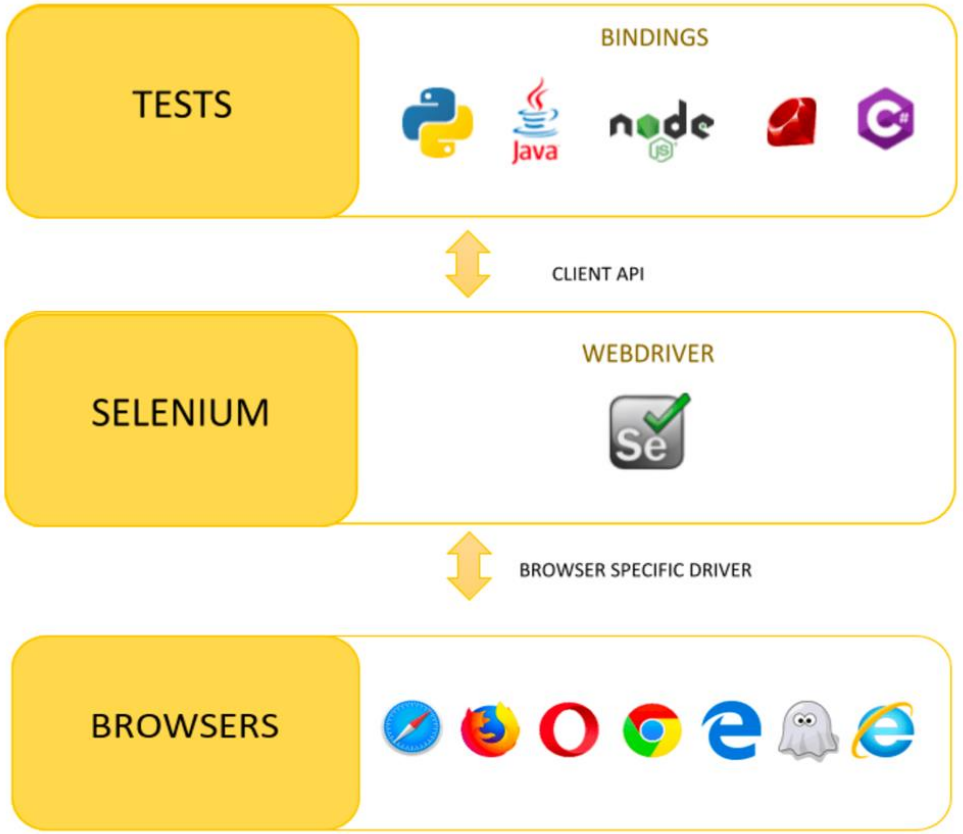
En capítulos anteriores...



https://github.com/oacarrillo/CursoVacacionalAutoNtt.git

Selenium

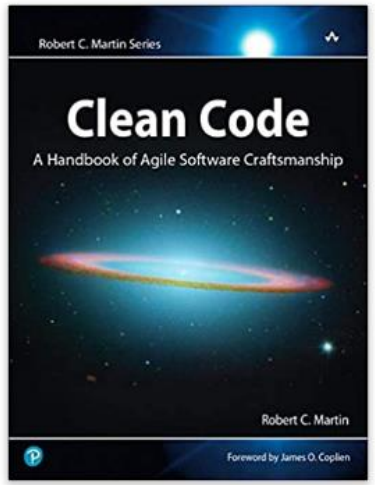
Selenium es un entorno de pruebas de software para aplicaciones basadas en la web. Selenium provee una herramienta de grabar/reproducir para crear pruebas sin usar un lenguaje de scripting para pruebas (Selenium IDE). Incluye también un lenguaje específico de dominio para pruebas (Selenese) para escribir pruebas en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, PHP y Python. Las pruebas pueden ejecutarse entonces usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.




>_
CODE

Principios FIRST

Libros › Computadoras y Tecnología › Programación



Escuchar

Ver esta imagen

Sigue a los autores

Robert C. Martin

Seguir

Clean Code: A Handbook of Agile Software Craftsmanship 1st Edición

de Robert C. Martin (Author)

4,703 calificaciones **4.4** en Goodreads 19,245 calificaciones

Parte de: Robert C. Martin (14 libros)

Ver todos los formatos y ediciones

Kindle	Pasta blanda
US$28.99	**US$42.10 - US$44.99**
Leer con nuestra **Aplicación gratuita**	7 Usado de US$39.31 2 Nuevo de US$44.99

Even bad code can function. But if code isn't clean, it can bring a development organization to its knees. Every year, countless hours and significant resources are lost because of poorly written code. But it doesn't have to be that way.

Noted software expert Robert C. Martin, presents a revolutionary paradigm with ***Clean Code: A Handbook of Agile Software Craftsmanship***. Martin, who has helped bring agile principles

Leer más

Reportar información de producto incorrecta

ISBN-10	ISBN-13	Edición	Editorial
9780132350884	978-0132350884	1er	Pearson

2008 - 2022

14 Años

Estructura de las pruebas unitarias

Por norma general, los unit test deberían seguir una estructura AAA, Este patrón fue creado por BILL WAKE (2001) y esta compuesto por:

•**Arrange** (Organizar): En esta parte de la prueba, debes establecer las condiciones iniciales para poder realizarla, así como el resultado que esperas obtener. Y esto significa por ejemplo, declarar las variables y crear las instancias de los objetos.

•**Act** (Accionar): Es la parte de ejecución, tanto del fragmento de código de la prueba, como del fragmento de código a testear.

•**Assert** (Comprobar o certificar): Por último, se realiza la comprobación para verificar que el resultado obtenido, coincide con el esperado.


```
[TestMethod]
public void TestMethod()
{
  //Arrange test
   testClass objtest = new testClass();
   Boolean result;

  //Act test
   result = objtest.testFunction();

  //Assert test
   Assert.AreEqual(true, result);
```

http://oscarmoreno.com/pruebas-unitarias/
https://www.youtube.com/watch?v=gPT6zhDGeww

Estructura de las pruebas unitarias

```java
public class Suma {
    public int sumar(int numeroUno, int numeroDos){
        return numeroUno + numeroDos;
    }
}
```

```java
@Test
public void sumarDosNumerosPositivos(){

    //Arrange
    Suma suma = new Suma();
    int numeroUno = 2;
    int numeroDos = 3;
    int resultadoObtenido;

    //Act
    resultadoObtenido = suma.sumar(numeroUno,numeroDos);

    //Assert
    Assert.assertEquals( expected: 5, resultadoObtenido);
}
```

Beneficios:

- Estándar (Mantenimiento)
- Se elimina la duplicidad
- Separación clara

1[Arrange

2[Act

3[Assert

Principios FIRST

Fast (rápido)

Independent (independiente)

Repeatable (repetible)

Self-validating (auto evaluable)

Timely (oportuno)

Principios FIRST - F

Fast (rápido)

Los test unitarios deben completar su ejecución lo más rápido posible. Al ser pruebas que se realizan (o deberían realizarse) sobre fragmentos pequeños de código, deben finalizar su ejecución lo antes posible.

https://images-wixmp-ed30a86b8c4ca887773594c2.wixmp.com/
https://media.tenor.com/7snjRBDh08oAAAAC/flash-slothmore-zootopia.gif

Principios FIRST - I

NTT DATA

Independent (independiente)

En las pruebas unitarias, el objeto de prueba no debe depender de otra unidad de código. Esto no significa que no requiera de datos para poder ejecutar ese código, pero esto se realizará a través de **mocks o stubs**, que vienen a ser «objetos falsos» creados específicamente para realizar las pruebas. Pero, en ningún caso, utilizaremos otras partes del código que no forme parte del unit test.

Dummy : Objetos ficticios que se pasan al método pero no se usan. Por lo general, solo se usan para rellenar listas de parámetros.
Fake : Objetos falsos que toman un atajo para cumplir suplir un valor de entrada o componente. Ej: Base de datos en memoria.
Stubs : Respuestas enlatadas para un solo contexto de prueba. Ej: Un parámetro constante (Cedula, Tipo)
Spies : Proporcionan información especifica reaccionando de acuerdo a como fueron llamados. Ej: Ingreso una cedula y me retorna # de líneas
Mocks : Objetos pre programados complejos (Subrutinas).

Cada prueba unitaria debe ser independiente de otra

- Se pueden ejecutar las pruebas en diferente orden
- El resultado no se ve alterado por el orden de ejecución de la prueba
- Su nombre lo dice prueba unitaria (No de integración)

https://martinfowler.com/articles/mocksArentStubs.html#TheDifferenceBetweenMocksAndStubs

Principios FIRST - R

Repeatable (repetible)

Las pruebas unitarias deben poder repetirse. De hecho, lo ideal es repetirla muchas veces. Por ejemplo, se deberían ejecutar las pruebas cada vez que suba un nuevo código al repositorio.

https://i.pinimg.com/736x/c7/69/08/c76908722b8fad727b92c92fa1a57b86.jpg

Repetible independiente del

- Servidor
- Computador
- Entorno
- No depende de configuración de usuario
- No depende de configuración de las herramientas
- Me deben dar el mismo resultado en cualquier entorno

Principios FIRST - S

Self-validating (auto evaluable)

Los unit test deben mostrar de forma clara, y sin que sea necesario tu intervención, el resultado de la prueba. Gracias a la parte de «arrange», donde estableces los resultados a obtener, la parte de «assert» puede responder si la prueba ha sido satisfactoria o no.

https://stock.adobe.com/es/images/ok-ko/17799019

Principios FIRST - T

Timely (oportuno)

Los test unitarios deben realizarse lo antes posible, el código no debe llegar a producción sin haber escrito y superado las pruebas unitarias. Incluso, si es posible, deberían desarrollarse antes que el código (Test Driven Development)

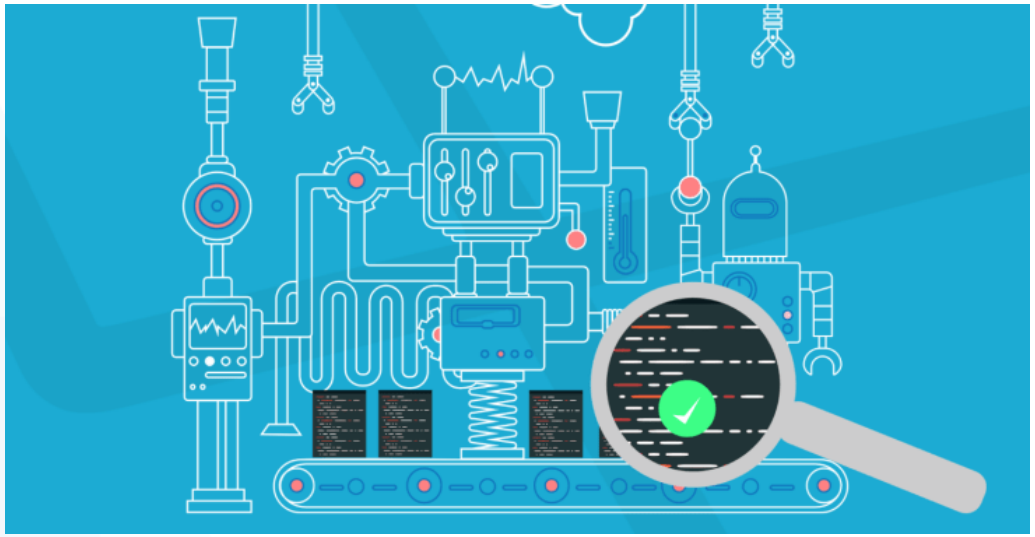
Cuando hacer o no pruebas unitarias

Se hacen para:

- Menos errores en producción
- Resolución mas rápida de errores
- Comprensión del código
- Sencillez de integración
- Mejorasen la estructura del código
- Localización de errores
- Reducción de costes

Cuando no es posible:

- El código cambia mucho
- El periodo de vida del producto es muy corto
- Componentes de menor importancia dentro del código con tiempos de entrega ajustados



Junit - Profundización

JUnit es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

JUnit es un conjunto de clases (*framework*) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

https://junit.org/junit5/docs/current/user-guide/


>_
CODE

Cobertura

¿Qué es la cobertura de pruebas unitarias?

La cobertura de pruebas unitarias es una métrica de QA que evalúa si los casos de prueba diseñados cubren el código de la aplicación y la cantidad de este código sometido a prueba cuando se ejecutan esos casos de prueba. Por lo tanto, la cobertura de pruebas ayuda a evaluar la efectividad de sus pruebas al ofrecer datos sobre varios elementos de cobertura.

Resumen:

- La cobertura es la medida que visibiliza porcentaje del código esta siendo ejecutado por pruebas automáticas
- Esta representado por un %
- Que un proyecto tenga pruebas unitarias, no es garantía que estés probando bien

https://alexandrade.net/blog-de-ingenieria-de-software/calidad-de-software/por-que-la-cobertura-de-pruebas-unitarias-es-una-parte-importante-de-qa/#.Y39513bMKiM

Criterios para Calculo de Cobertura (1)

1) Cobertura del producto: Como su nombre lo indica, la cobertura del producto no es más que una cobertura de pruebas desde la perspectiva del producto. En otras palabras, es preguntarse sobre qué áreas del producto ha probado.

Ejemplo: Supongamos que se debe probar una aplicación simple como una calculadora. Aunque debe verificar las funciones esenciales como las cuatro operaciones aritméticas, eso no es suficiente. También debe considerar otros factores al probar la aplicación de la calculadora. Hay un esfuerzo adicional en las pruebas de varios escenarios, como lo bien que la calculadora maneja grandes cantidades. O, ¿qué pasa si el usuario hizo algo inusual como pegar caracteres especiales en el campo de texto?

2) Cobertura de riesgos: La cobertura de riesgos consiste en evaluar los riesgos involucrados en una aplicación y probarlos en detalle. Es necesario enumerar todos los posibles riesgos que pueden ocurrir en la aplicación y verificarlos adecuadamente.

Ejemplo: en una aplicación de entrega de alimentos, el usuario elige el restaurante y las preferencias de comida y paga a través de la pasarela de pago integrada. Un riesgo común es que los usuarios se desconecten cuando están en el proceso de pago. ¿Cómo se comportará la aplicación bajo este escenario? Asimismo, se deben considerar otros factores de riesgo relevantes que intervienen en la aplicación y comprobarlos.

3) Cobertura de valor límite: El análisis del valor límite es una parte de las pruebas de software que permite al *tester* crear los casos de prueba necesarios para un campo de entrada.

Ejemplo: un campo de entrada numérico solo debe permitir valores del 0 al 50. Por lo tanto, puede probar la aplicación ingresando números menores que 0, como negativos, y números mayores a 50; así como verificar que no se permitan escribir caracteres diferentes a números.

https://alexandrade.net/blog-de-ingenieria-de-software/calidad-de-software/por-que-la-cobertura-de-pruebas-unitarias-es-una-parte-importante-de-qa/#.Y39513bMKiM

Criterios para Calculo de Cobertura (2)

4) Cobertura de requisitos: La cobertura de los requisitos es la más crucial de todas las técnicas discutidas en este artículo. Por supuesto, incluso si su aplicación está libre de errores y funciona bien, ¿qué pasa si no cumple con los requisitos del usuario? Esto no solo pone en riesgo al proyecto, sino que afecta fuertemente la imagen de la empresa. Por eso el llamado a tener un plan de pruebas que valide que la aplicación cumpla con todos los requisitos. Después de todo, esa es toda la lógica detrás del desarrollo de software.

5) Automatización de pruebas asistida por IA: La automatización de pruebas asistida por IA (Inteligencia Artificial) es la técnica de cobertura de pruebas más avanzada de todas las mencionadas anteriormente. Esta utiliza herramientas de automatización que hacen que su enfoque de prueba pase al siguiente nivel. Estas herramientas asistidas por IA vienen con conjuntos de pruebas que aplican el aprendizaje automático (ML: *Machine Learning*) para aprender con cada ejecución. La IA incluso permite la autocorrección de casos de prueba, lo que alivia la carga del mantenimiento de la prueba; ayudando a obtener un sólido conjunto de pruebas que proporciona una cobertura de pruebas *premium*.

https://alexandrade.net/blog-de-ingenieria-de-software/calidad-de-software/por-que-la-cobertura-de-pruebas-unitarias-es-una-parte-importante-de-qa/#.Y39513bMKiM

¿Por que es importante la cobertura?

- Encontrar y rectificar errores en etapas tempranas
- Crear más casos de prueba para garantizar una mejor cobertura
- Eliminar casos de prueba no deseados
- Posea un mejor control sobre su proyecto
- Ciclos de prueba eficientes
- Mayor ROI

¿Como calcular la cobertura de pruebas unitarias?

```
Cobertura de pruebas = (Número de líneas que se han probado / Número total de líneas de código de la aplicación) x 100
```

Si una aplicación que tiene 1000 líneas de código y 400 de ellas son ejecutadas por al menos un caso de prueba; su cobertura de pruebas es del 40%.

https://vipinrajnair.medium.com/why-is-test-coverage-an-important-part-of-software-testing-f7ac168a6d90


NTT DATA
GRACIAS
FUTURE
AT HEART