Proyecto – Pruebas de Cobertura

Fidel Lace

Luigui Salazar

Pardo Aliaga

Análisis y diseño Orientado a Objetos

Profesor: Eric Gustavo Coronel Castillo

Universidad Científica del Sur

2017

Contenido

1. Objetivo

El objetivo principal de este trabajo es presentar un ejemplo de gestión de un proyecto de testing de software. Para ello hemos requerido plasmar algunas experiencias de diferentes proyectos que se encuentran en la web, creando una guía o referencia para el desarrollo de este tipo de ofertas.

Poniendo más en claro los puntos de este proyecto, está divido en dos partes. La primera parte es una exposición de los fundamentos del testing. Conceptos básicos, herramientas, equipos que intervienen, metodologías... Se podría entender como las partes más importantes que cualquier persona que se dedique al testing debe tener claro para desarrollar las tareas propias de este mundo, ya sea ingeniero junior y no digamos un manager. La segunda parte es la presentación de un caso práctico de la gestión de un proyecto.

2. Introducción

- “Software testing es una metodología para encontrar defectos en el software”
- “Software testing es el proceso utilizado para medir la calidad de cualquier software desarrollado”
- “Software testing es cualquier actividad dirigida a evaluar una característica o capacidad de un programa y determinar si se cumplen los requisitos deseados” “Testing es el proceso de verificación del correcto funcionamiento de una aplicación”

En pocas palabras el término testing de software es aquella prueba que se encarga de encontrar las posibles diferencias en cuanto al funcionamiento deseado de un determinado software y su funcionamiento real.

¿Por qué el software usualmente no funciona correctamente? Debido a los errores de las personas. Las personas encargadas de desarrollar la aplicación, pueden cometer errores en su diseño y desarrollo. Esos errores hacen que aparezcan errores inherentes a la aplicación. A estos errores se les denomina defectos o bugs.

3. ¿Qué es la cobertura?

La cobertura es la cantidad de código (medida porcentualmente) que está siendo cubierto por las pruebas. O sea, ejecuto las pruebas de mi aplicación y si hay alguna línea de mi código que nunca fue ejecutada en el contexto de las pruebas, entonces dicha línea no está cubierta. Si mi código consta de 100 líneas y solo 50 líneas están siendo ejecutadas al correr las pruebas, entonces mi cobertura es del 50%.

También podemos decir que la cobertura de código es la cantidad de código que está sometido a nuestras pruebas. A mayor cobertura mayor cantidad de código está siendo probado por nuestras pruebas unitarias.

4. ¿Qué beneficio tiene medir la cobertura?

Una respuesta genérica podría ser que aumenta la calidad de mi aplicación. Siendo más concreto podría decir que si tengo una alta cobertura, significa que gran parte me mi código está siendo probado y por consiguiente podría tener cierta

certeza sobre el correcto funcionamiento de mi aplicación. Al mismo tiempo medir la cobertura podría ayudarme a detectar código innecesario en mi aplicación, ya que es código que no se ejecuta.

5. ¿Una cobertura del 100% asegura que mi código no tiene bugs?

De ninguna manera, una cobertura del 100% solo nos dice que todo nuestro código está siendo cubierto por pruebas, pero puede que las pruebas no estén contemplando algunas situaciones, o sea, que faltan pruebas o incluso podría ocurrir que las pruebas fueran deficientes.

6. Pruebas unitarias

Un unit test es un método que prueba una unidad de código. Al hablar de una unidad de código nos referimos a un requerimiento. Muchos desarrolladores tienen su propio concepto de lo que es una prueba unitaria; sin embargo, la gran mayoría coincide en que una prueba unitaria tiene las siguientes características:

- Prueba solamente pequeñas cantidades de código: Solamente prueba el código del requerimiento específico.
- Se aísla de otro código y de otros desarrolladores: El unit test prueba exclusivamente el código relacionado con el requerimiento y no interfiere con el trabajo hecho por otros desarrolladores.
- Solamente se prueban los endpoints públicos: Esto principalmente porque los disparadores de los métodos privados son métodos públicos por lo tanto se abarca el código de los métodos privados dentro de las pruebas.

- Los resultados son automatizados: Cuando ejecutamos las pruebas lo podemos hacer de forma individual o de forma grupal. Estas pruebas las hace el motor de prueba y los resultados de los mismos deben de ser precisos con respecto a cada prueba unitaria desarrollada
- Repetible y predecible: No importa el orden y las veces que se repita la prueba, el resultado siempre debe de ser el mismo.
- Son rápidos de desarrollar: Contrariamente a lo que piensan los desarrolladores –> que el desarrollo de pruebas unitarias quita tiempo – los unit test por lo general deben de ser simples y rápidos de desarrollar. Difícilmente una prueba unitaria deba de tomar más de cinco minutos en su desarrollo.

7. Pruebas de integración

Pruebas integrales o pruebas de integración son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias y lo que prueban es que todos los elementos unitarios que componen el software, funcionan juntos correctamente probándolos en grupos. Sus características principales son:

- Se deben ensamblar o integrar los módulos para formar el paquete de software completo.
- La prueba de integración se dirige a todos los aspectos asociados con el doble problema de verificación y de construcción del programa.

- Las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra.

8. Pruebas funcionales

Una prueba funcional es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Las pruebas funcionales se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático. Dicho de otro modo son pruebas específicas, concretas y exhaustivas para probar y validar que el software hace lo que debe y sobre todo, lo que se ha especificado.

9. Tecnologías de testing

- ***Frameworks** de **test** unitarios y de integración : **JUnit***
- ***Mocking** (creación de objetos simulados) : **Mockito***
- ***Framework** de **matchers** (facilita comprobaciones) : **Hamcrest***
- ***Frameworks** de **test** funcionales : **Selenium***
- ***Navegador** (navegador sin interfaz grafica basado en Webkit): **Phantom.is***
  - WebDriver para PhantomJS, : Ghost Driver
- ***Wrapper** de **Selenium** : **Fluentlenium***


10. Conclusiones

No hay que buscar la calidad perfecta ni el 100% de cobertura, esto no es inteligente ni práctico, ya que nos llevaría demasiado tiempo y esfuerzo. Pero si son necesario unos mínimos de calidad y enfocar nuestros esfuerzos a probar las piezas más complicadas o más importantes para negocio.

Herramientas como Cobertura nos ayudan enormemente a conseguir estos objetivos y son un aliado fundamental para entornos de mejora continua.

Recordar siempre que, sin medir, es imposible mejorar. Hay que medir antes y después, y comparar las medidas. Eso es lo que realmente me indica si estoy mejorando o empeorando.

BIBLIOGRAFIA:

- https://es.wikipedia.org/wiki/Cobertura_de_c%C3%B3digo
- https://www.adictosaltrabajo.com/tutoriales/maven-cobertura/#10.%20Conclusiones%7Coutline
- file:///C:/Users/lsalazar.ITECH/Downloads/expoqa2016coberturafinal-160609160113.pdf
- https://msdn.microsoft.com/es-es/communitydocs/alm/unit-test
- http://blog.abstracta.com.uy/2015/04/medir-la-cobertura-de-pruebas-unitarias.html