**Proyecto – Pruebas de Cobertura**

**Fidel Lace  
Luigui Salazar**

**Daniel Pardo**

Análisis y diseño Orientado a Objetos

Profesor: Eric Gustavo Coronel Castillo

Universidad Científica del Sur

2017

**Contenido**

[1. Objetivo 3](#_Toc485461212)

[2. Introduccion 3](#_Toc485461213)

[3. ¿Qué es la cobertura? 4](#_Toc485461214)

[4. ¿Qué beneficios tiene medir la cobertura? 4](#_Toc485461215)

5. [**¿**Una cobertura del 100% asegura que mi código no tiene bugs? 5](#_Toc485461216)

[6. Pruebas unitarias 5](#_Toc485461217)

[7. Pruebas de integracion 6](#_Toc485461217)

[8. Pruebas funcionales 8](#_Toc485461217)

[9. Tecnologias testing 8](#_Toc485461217)

10. [Referencias 9](#_Toc485461223)

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.

1. **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.

1. **¿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.

1. **¿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.

1. **¿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.

1. **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.

Las pruebas unitarias tienen que poder ejecutarse independientemente del estado del entorno. Las pruebas tienen que pasar en cualquier ordenador del equipo de desarrollo.

La ejecución de una prueba no puede afectar la ejecución de otra. Después de la ejecución de una prueba el entorno debería quedar igual que estaba antes de realizar la prueba.

Las diferentes relaciones que puedan existir entre módulos deben ser simulada para evitar dependencias entre módulos.

Es importante conocer claramente cuál es el objetivo del test. Cualquier desarrollador debería poder conocer claramente cuál es el objetivo de la prueba y su funcionamiento. Esto sólo se consigue si se trata el código de pruebas como el código de la aplicación.

Es importante tener en cuenta que aunque estas son las características de una buena prueba, no siempre será posible ni necesario cumplir con todas estas reglas y será la experiencia la que nos guiará en la realización de las mismas

**Beneficios de las pruebas unitarias**Con las pruebas unitarias todos ganan. La vida de desarrollador será mucho más fácil, ya que la calidad de su código mejorará, se reducirán los tiempos de depuración y la corrección de incidencias y por tanto el cliente estará mucho más contento porque la aplicación hace lo que él quiere que haga, por lo que ha pagado.

Las pruebas fomentan el cambio y la refactorización. Si consideremos que nuestro código es mejorable podemos cambiarlo sin ningún problema. Si el cambio no estuviera realizado correctamente las pruebas nos avisarán de ello. Seguramente la frase “si funciona no lo toques” a más de uno familiar les resultará familiar. Si hubiera pruebas unitarias, no sería necesario pronunciarla.

Se reducen drásticamente los problemas y tiempos dedicados a la integración. En las pruebas se simulan las dependencias lo que nos permite que podemos probar nuestro código sin disponer del resto de módulos. Por experiencia puede decir que los procesos de integración son más de una vez traumáticos, dejándolos habitualmente para el final del proyecto. La frase “sólo queda integrar” haciendo referencia a que el proyecto está cerca de terminar suele ser engañosa, ya que el periodo de integración suele estar lleno de curvas.

Las pruebas nos ayudan a entender mejor el código, ya que sirven de documentación. A través de las pruebas podemos comprender mejor qué hace un módulo y que se espera de él.

Nos permite poder probar o depurar un módulo sin necesidad de disponer del sistema completo. Aunque seamos los propietarios de toda la aplicación, en algunas situaciones montar un entorno para poder probar una incidencia es más costoso que corregir la incidencia propiamente dicha. Si partimos de la prueba unitaria podemos centrarnos en corregir el error de una forma más rápida y lógicamente, asegurándonos posteriormente que todo funciona según lo esperado.

**Mitos de las pruebas unitarias**

Aunque los beneficios de las pruebas unitarias puedan parecer claros, no es más cierto que a día de hoy se usan en muy pocos proyectos. Pero sin son tan buenas,¿por qué no se usan? Pues la razón principal es que existen bastante desconocimiento en esta materia, poca tradición y algunos falsos mitos.

Uno de los mitos es creer que escribir pruebas unitarias es escribir el doble de código; escribir el código de la aplicación y escribir el código de pruebas. Escribir una prueba nunca es escribir el doble de código, aunque lógicamente sí es escribir más código. El mito es totalmente falso.

Las pruebas siempre se deben ir escribiendo a medida que se desarrolla el software. A medida de desarrollamos vamos probando nuestro código lo que nos permite asegurar que el módulo queda terminado correctamente, libre de incidencias. La realización de pruebas unitarias debe ser un proceso obligatorio en nuestros desarrollos y que no queden a la voluntad del desarrollador. Si el desarrollador no está habituado a su uso diario es muy fácil que tienda a evitar realizar este tipo de pruebas.

Si no estás familiarizado con el uso de pruebas unitarias la perseverancia debe ser tu gran aliado. Debes estar convencido de que el uso de pruebas unitarias es el enfoque correcto y no ser flexible; las pruebas unitarias no son opcionales y deben realizarse a medida que se desarrolla. Dejar la implementación de pruebas para el final no es realista, ya que las pruebas nunca se llegarán a implementar y si lo hacen, serán de una baja calidad, ya que la persona que las realiza considerará que es una pérdida de tiempo, considerando que su tarea ya estaba terminada sin realizar estas pruebas.

**¿Y si no escribiéramos este código cómo probamos?**

Hay diferentes situaciones. La primera situación es que no se prueba. Se implementa el módulo pero no se prueba, dando por hecho que nuestra experiencia como desarrolladores hará que las incidencias sean mínimas. Cuando tenemos todos los módulos lo probamos de forma integrada a través de pruebas funcionales.

Esta situación es la situación peor con la que podemos encontrarnos, pero desgraciadamente es la más habitual. Los tiempos de integración, depuración y corrección de incidencias se multiplican. El tiempo que va desde que consideramos que el producto está terminado hasta que realmente está terminado puede llegar a ser superior al tiempo invertido en el desarrollo, y lo digo por experiencia, por mala experiencia.

Otra situación algo mejor, es el uso de aplicaciones de pruebas de usar y tirar. Todos hemos implementado aplicaciones tontas, generalmente de consola, para probar el módulo que estamos desarrollando y que posteriormente integraremos en un sistema mayor. La situación no es la ideal pero realmente el desarrollo de pruebas unitarias puede tener una semejanza con este tipo de aplicaciones, siendo éste último concepto más avanzado y mejorado para que pueda ser reutilizable, independientes y completas.

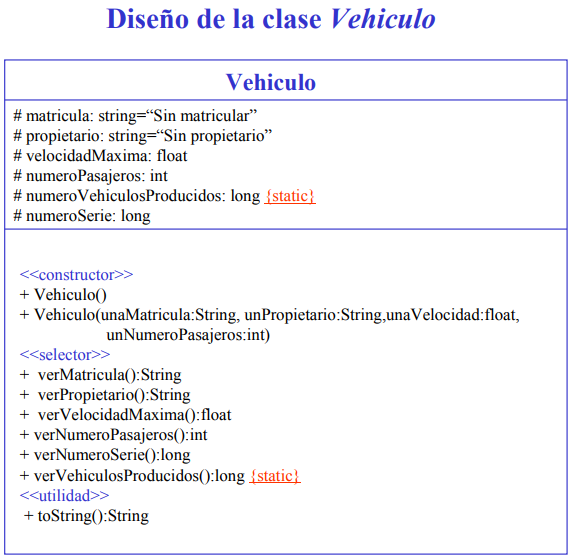
Otro mito, unido al anterior, es que desarrollar pruebas unitarias hace que los tiempos de desarrollo se incrementen, incrementando los costes del proyecto. Entre desarrollar y no probar y desarrollar haciendo pruebas unitarias, reconoceré que es más rápido desarrollar sin probar…..pero creo que esto no nos vale….así que entre desarrollar con pruebas unitarias y desarrollar probando con métodos más rústicos, ésta segunda es más lenta y por tanto, más cara.

La manera más rápida de desarrollar software es desarrollarlo bien. Como hemos comentado en el punto anterior, si no desarrollamos código de calidad y no realizamos pruebas, los tiempos de desarrollo se podrán disparar enormemente; tiempo de integración, tiempo de depuración, tiempo de incidencias etc… Pero hay un problema mayor; la pérdida de confianza del cliente. Un producto de baja calidad, lleno de incidencias es la peor tarjeta de presentación. Por otro lado, hay que tener en cuenta que cuando más larga sea la vida de la aplicación más beneficios se obtendrán del sistema de pruebas unitarias.

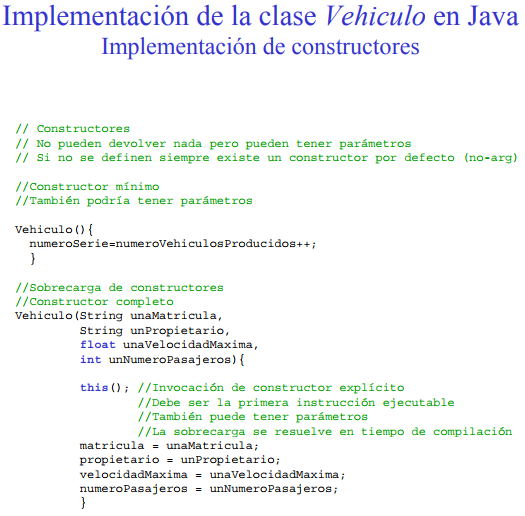
Muchos productos tienen un mantenimiento evolutivo que implica el desarrollo de nuevas funcionalidad y la corrección de incidencias. Sin un proceso adecuado de pruebas unitarias, los tiempos de pruebas se dispararán en fases posteriores, ya que será necesario probar y corregir tanto las nuevas funcionalidades como las funcionalidades ya existentes.

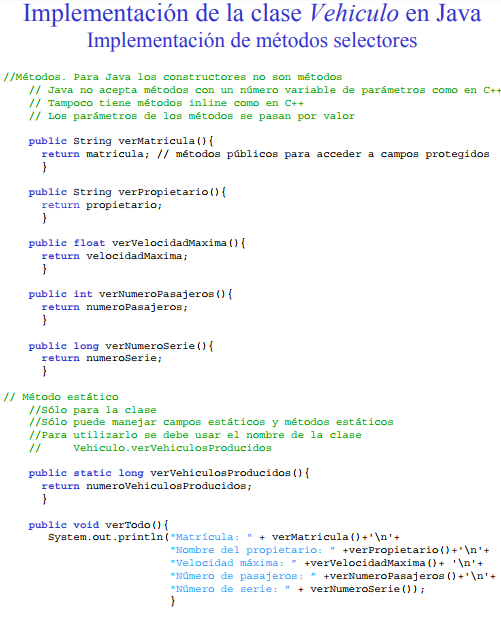
Cierto es también que la adopción de pruebas unitarias dentro del equipo de desarrollo implica una inversión en formación. La primera vez que las empleemos llevará algo más de tiempo ya que tenemos que familiarizarnos con las herramientas y con el framework de pruebas unitarias. Este tiempo es algo habitual asociado a cualquier proceso de aprendizaje que sólo sería imputable al primer proyecto que se desarrolle.

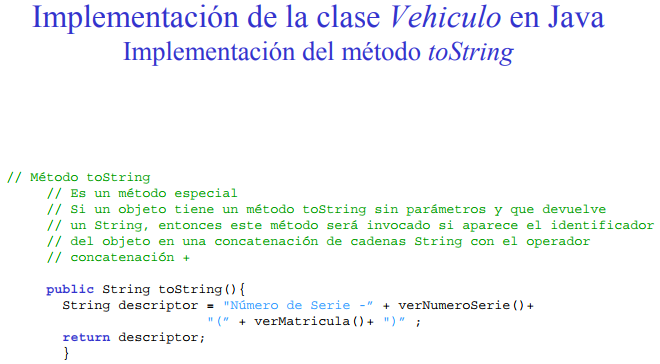
**Ejemplo Java**

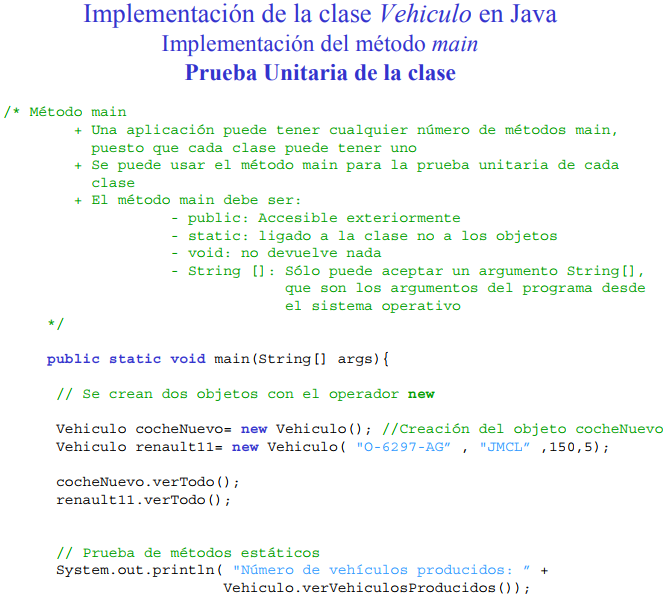


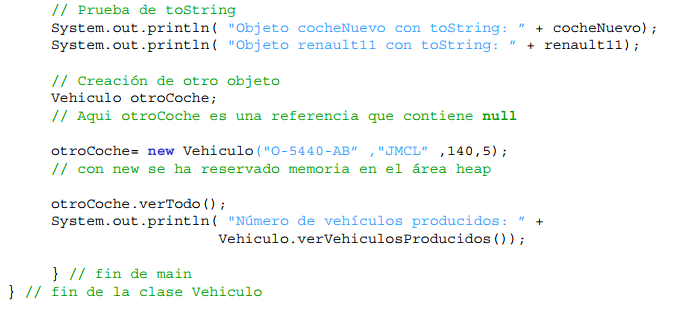












1. **Pruebas de integración**

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:

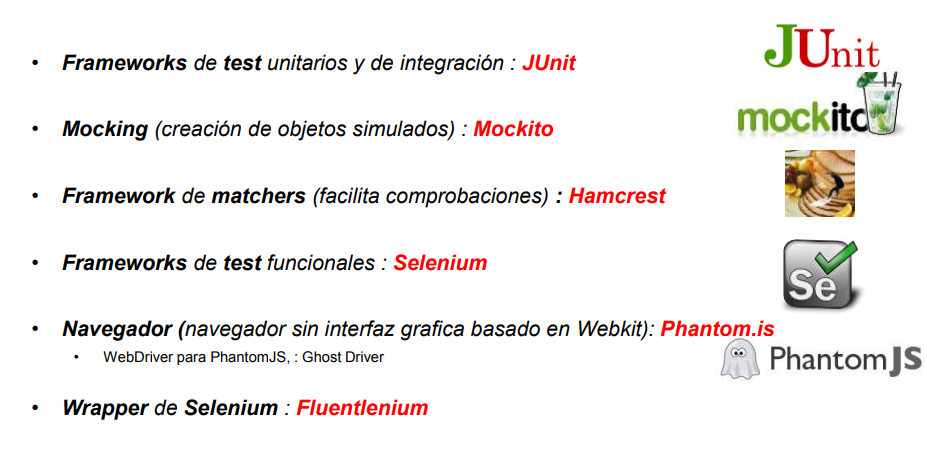
* Se deben ensamblar o integrar los módulos para formar el paquete de software completo.
* La prueba de integración se dirige a todos los aspectos asociados con el doble problema de verificación y de construcción del programa.
* Las técnicas que más prevalecen son las de diseño de casos de prueba de caja negra.

1. **Pruebas funcionales**

Una prueba funcional es una prueba basada en la ejecución, revisión y retroalimentación de las funcionalidades previamente [diseñadas](https://es.wikipedia.org/wiki/Dise%C3%B1o_orientado_a_objetos) para el [software](https://es.wikipedia.org/wiki/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](https://es.wikipedia.org/wiki/Inform%C3%A1tica). 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.



1. **Tecnologías de testing**



1. **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>
* <https://si.ua.es/es/documentacion/csharp/documentos/pruebas/07pruebasunitarias.pdf>
* <http://mmc.geofisica.unam.mx/acl/Herramientas/Analisis_y_Diseno_Orientado_a_Objetos.pdf>
* <http://tesis.pucp.edu.pe/repositorio/bitstream/handle/123456789/1562/ROMERO_GALINDO_RAUL_SISTEMA_INFORMACION_EDUCACION_ESPECIAL.pdf?sequence=1>