Implementación de Herramientas de Software para mejorar la Aplicación de Pruebas Unitarias en la Etapa de Construcción del Proceso de Desarrollo y Mantenimiento de Software de la Norma NMX-I-059-NYCE-2011 (MOPROSOFT)

Adriana de la Roca, Leticia Santa, Angel Estrada, Boris Aranda, y Laura Villavicencio

A. de la Roca, L. Santa, A. Estrada, B. Aranda y L. Villavicencio
Instituto Tecnológico de Zacapetec.Av. Tecnológico No. 27, Colonia Centro, Zacatepec, Morelos
Universidad Tecnológica Emiliano Zapata del Estado de Morelos. México

M. Ramos., V.Aguilera., (eds.) . Ciencias de la Ingeniería y Tecnología, Handbook -©ECORFAN- Valle de Santiago, Guanajuato, 2014.

Abstract

In recent times, the need to improve the quality of processes and products in the software industry has led to the development and implementation of standards and models to guide companies in each of the stages of software development. In this investigation the Process of Software Development and Maintenance of Category Operation NMX-I-059/02-NYCE-2011 standard "- Software Engineering - Information Technology Product Quality" addresses (MoProSoft) and the implentación software tools to improve the implementation of unit testing in this process. It is noteworthy that chose the Java programming language and free software tools in order to investigate and explain the use of tools for unit testing.

10 Introducción

Hoy en día la calidad del software es un tema preocupante para las industrias del desarrollo del software. Las industrias tienen la exigencia de contar con un modelo o estándar que les permita normalizar y verificar todos sus procesos con el fin de mejorar dia con dia. En México actualmente más de 300 empresas se encuentran certificadas bajo la norma NMX-I-059-NYCE- 2011(MoProSoft). [1] El Centro de Desarrollo de Software de la Universidad Tecnológica Emiliano Zapata (CDS- UTEZ), se encuentra dentro de la empresas mexicanas que trabajan bajo dicha norma. Actualmente se encuentran en el Nivel 3 de madurez [2], y buscando siempre la mejora y excelencia en sus procesos.

El origen de MoProSoft es la necesidad de cumplir con la estrategia número 6 del Programa de Software (ProSoft) de la Secretaría de Economía (establecida desde el sexenio 2000-2006), con el propósito de alcanzar niveles internacionales en el desarrollo de procesos principalmente por parte de las pequeñas y medianas empresas que se dedican a la industria del software. Como empresa el contar con un estándar y además estar certificados permitirá tener mayor penetración en el mercado, local, regional, nacional e internacional. [3]

De acuerdo al NYCE [4], la verificación del proceso de desarrollo de software es importante debido a que existen puntos clave que son de beneficio para los usuarios de las organizaciones dedicadas al desarrollo y mantenimiento de software como son:

- Certidumbre, ya que las empresas verificadas deben llevar a cabo sus actividades con prácticas validadas por las normas mexicanas.
- Calidad, por que al llevar a cabo buenas prácticas de desarrollo y mantenimiento de software los resultados son medibles.
- Capacidad, ya que los procesos con los que se desarrolla o mantiene el software son repetibles.

El modelo de procesos (MoProSoft) tiene tres categorías de procesos: Alta Dirección, Gestión y Operación que reflejan la estructura de una organización. [5]

- La categoría de Alta Dirección contiene el proceso de Gestión de Negocio.
- La categoría de Gestión está integrada por los procesos de Gestión de Procesos, Gestión de Proyectos y Gestión de Recursos. Éste último está constituido por los subprocesos de Recursos Humanos y Ambiente de Trabajo, Bienes, Servicios e Infraestructura y Conocimiento de la Organización.

- La categoría de Operación está integrada por los procesos de Administración de Proyectos Específicos y de Desarrollo y Mantenimiento de Software.

Esta investigación se enfoca en el proceso de Desarrollo y Mantenimiento de Software de la categoría de Operación. El propósito de Desarrollo y Mantenimiento de Software es la realización sistemática de las actividades de análisis, diseño, construcción, integración y pruebas de productos de software nuevos o modificados cumpliendo con los requerimientos especificados.[5]

El desarrollo y mantenimiento de software se lleva a cabo a través de esta serie de actividades realizadas por equipos de trabajo. Dentro de estas actividades se maneja a las pruebas de integración como una etapa siguiente a la codificación. En esta investigación se analizan herramientas de software que permiten mejorar la aplicación de pruebas unitarias de manera continua utilizando el modelo de Integración Continua, el fin de conocer y mejorar la calidad de nuestro software durante el desarrollo del mismo, a través de los informes generados por las herramientas que utilizan diferentes métricas de calidad.

Es importante mencionar que la correcta aplicación de pruebas unitarias en el modelo de Integración Continua, facilita e influye de manera directa a las pruebas de integración debido a que se asegura que las dos etapas se concluyan con tiempo estimado para entregar el producto.

Pruebas Unitarias

Una prueba unitaria es una pieza de un código (generalmente un método) que invoca otro pedazo de código y comprueba la exactitud de algunas suposiciones después. [6] Una prueba unitaria está caracterizada por probar la funcionalidad lógica y estructural de cada método o función excluyendo a los demás componentes, es decir, toma a cada método como una unidad independiente de los demás.

Tipologías

- Enfoque estructural o caja blanca: Se verifica la estructura interna del componente con independencia de su funcionalidad.
- Enfoque funcional o caja negra: Se comprueba el correcto funcionamiento de los componentes analizando las entradas y las salidas, verificando que el resultado es el esperado.

El objetivo principal de las pruebas unitarias es asegurar que el código que se ha programado cumple con la tarea establecida y contiene el mínimo de errores.

Beneficio de las pruebas unitarias

Los beneficios que se obtienen al aplicarse las Pruebas Unitarias dentro del desarrollo del software, así como la importancia de que esto se lleve a cabo son los siguientes:

Reducción de problemas y tiempos dedicados a la integración

En las pruebas unitarias los métodos deben probarse independientemente de los otros. En caso de que exista una dependencia, esta podrá simularse.

Mejorar la documentación

Como consecuencia de aplicar las pruebas unitarias, debe existir una mejor documentación para comprender la funcionalidad del método.

Probar sin un sistema completo

Nos permite poder probar o depurar un módulo sin necesidad de disponer del sistema completo.

Mejora la calidad del código

El software desarrollado presentará una reducción de errores que posiblemente afectarían en las etapas finales de desarrollo. Se reduciría los tiempos de depuración y la corrección de incidencias. Para poder realizar esto se busca encontrar partes de código que puedan:

- Reducir el rendimiento.
- Provocar errores en el software.
- Complicar el flujo de datos.
- Tener una excesiva complejidad.
- Suponer un problema en la seguridad.

Propiedades de las pruebas unitarias

Según Roy Osherove [6], una prueba unitaria debe tener las siguientes propiedades:

- Deber ser automatizada.
- Deben ser repetibles.
- Cubrir la totalidad del código
- Ejecución independiente.
- Relaciones simuladas.
- Objetivo claro.

Las pruebas unitarias en la Integración Continua

Integración continúa

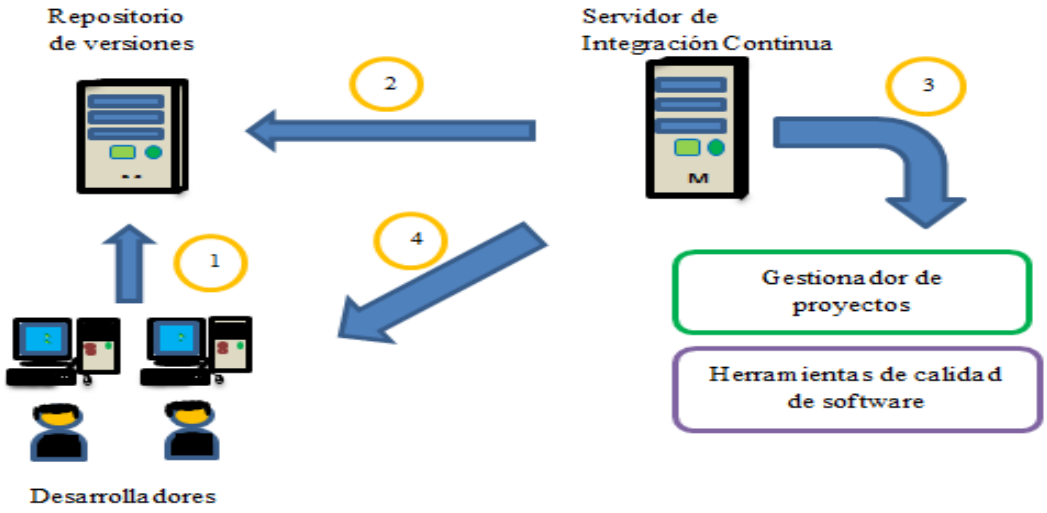
La integración continua es una práctica de desarrollo de software en la cual los miembros de un equipo integran su trabajo frecuentemente, como mínimo de forma diaria. Cada integración se verifica mediante una herramienta de construcción automática para detectar los errores de integración tan pronto como sea posible (Martin Fowler,2006).

Arquitectura de la Integración Continúa

La aplicación de pruebas unitarias puede realizarse de forma automatizada y continua durante todo el desarrollo del proyecto, permitiendo conocer a través de los resultados de las pruebas, la calidad de nuestro proyecto. Para ello se requiere la implementación de las herramientas de calidad de software dentro de la metodología Integración Continua.

En la figura se describe la arquitectura y el procedimiento que conlleva la integración continúa.

Figura 10 Arquitectura de la Integración Continua



1.- Los desarrolladores suben su código y sus pruebas unitarias de caja negra que son elaboradas utilizando algún framework a un controlador de versiones.

2.- El servidor de Integración Continua detecta los cambios en el repositorio y ejecuta el proceso de acciones pre configuradas, que corresponden al desarrollo del software.

3.- El gestionador de proyectos es el encargado de la compilación del proyecto y la ejecución de pruebas funcionales y estructurales.

Para que se pueda realizar esto, las herramientas de calidad de software deben estar instaladas junto al gestionador de proyectos.

4.- Los resultados del procedimiento se muestran a los desarrolladores y se libera la última versión del proyecto, es decir, cada componente que pase la prueba se guarda en otra rama del repositorio para aplicar las pruebas de integración, asegurando que cumplen con la funcionalidad prevista. El servidor de integración continua es el encargado de mostrar dichos resultados.

Herramientas de calidad de software

Las herramientas de calidad de software permiten ejecutar las pruebas unitarias de forma automatizada, obteniendo como resultado un informe donde se describen los errores presentados, así como las métricas suficientes para determinar la calidad del software.

Las herramientas que se presentan a continuación son destinadas a pruebas unitarias en el lenguaje de programación Java.

Tabla 10 Herramientas de calidad de software para pruebas unitarias en lenguaje Java

Nombre	Tipo de prueba	Tipo de licencia	Lenguajes soportados
JUnit	Funcional	Common Public License	Java
PMD	Estructural	BSD-style license.	Java, C, C + +, C #, PHP, Ruby, Fortran, JavaScript, entre otros.
Checkstyle	Funcional	GNU Lesser General Public License	Java
FindBugs	Estructural	GNU Lesser General Public License	Java
SonarQube	Estructural	LGPL v3	Java, C#,C/C++,PL/SQL,Cobol,PHP, entre otros.

JUnit

JUnit es un simple framework para escribir pruebas repetibles. [7] JUnit permite realizar pruebas unitarias de caja negra, es decir, evalúa el funcionamiento de cada uno de los métodos para determinar si su comportamiento es el esperado. JUnit se basa en otorgar al método un valor de entrada y en función al valor de retorno determinar si el método cumple con las especificaciones dadas. El método será exitoso en caso de que el valor de retorno esperado sea el indicado para dicho método, o en caso contrario JUnit devolverá un fallo en el método correspondiente.

PMD

PMD es un analizador de código estático. Encuentra defectos comunes de programación.[8] El análisis estático de código consiste en un proceso en donde sin necesidad de ejecutarse se evalúa. El analizador estático de código, recibe un código fuente, lo procesa y nos arroja una serie de sugerencias.

PMD revisa el código fuente y busca problemas potenciales como:

- Posibles errores: instrucciones try/catch/finally/switch vacías.
- Código muerto: variables locales, parámetros y métodos privados no usados
- Código sub óptimo: excesivo uso de String/StringBuffer
- Expresiones complicadas: instrucciones if innecesarias para ciclos que pueden ser ciclos while.
- Código duplicado: copiar y pegar código significa hacerlo también con los errores.

Checkstyle

Checkstyle es una herramienta de desarrollo para ayudar a los programadores a escribir código Java que se adhiere a un estándar de codificación.[9]

Checkstyle permite comprobar, por ejemplo:

- Javadoc comentarios para las clases, atributos y métodos
- Convenciones de nombres de atributos y métodos
- Limitar el número de parámetros de la función, las longitudes de línea
- Presencia de cabeceras obligatorias
- El uso de paquetes de las importaciones, de las clases, de los modificadores de alcance y de instrucciones de bloques
- Los espacios entre algunos personajes
- Las buenas prácticas de construcción de clase

FindBugs

FindBugs al igual que PMD es una herramienta que realiza un análisis estático de código en busca de errores potenciales en programa escritos en código Java. FindBugs opera sobre el bytecode Java y no sobre el código fuente.

SonarQube

SonarQube es una aplicación destinada para realizar pruebas de calidad de código. Esta herramienta permite hacer un análisis estático del código y mostrar los resultados en un ambiente gráfico comprensible. SonarQube integra a PMD, FindBugs y Checkstyle como analizadores de código. SonarQube es el lugar central para administrar la calidad de código, ofreciendo reportes visuales a través de proyectos y habilitando la reproducción de un análisis anterior para seguir las métricas de evolución. [10]

Sonarqube cubre los 7 aspectos de calidad de código: [11]

- Arquitectura y diseño
- Duplicaciones
- Pruebas unitarias
- Complejidad
- Errores potenciales
- Reglas de codificación
- Comentarios

Analisis de la situacion actual

En el CDS-UTEZ se desarrolla un proyecto de Sistema Integral de Servicios Académicos. El Equipo de Trabajo de acuerdo con la NORMA NMX-I-059/03-NYCE-2005, esta conformado por 9 roles que involucra: Responsable de Administración del Proyecto, Responsable de Desarrollo y Mantenimiento de Software, Analista, Diseñador de Interfaz de Usuario, Diseñador, Programadores , Responsable de Pruebas, Revisor y Responsable de Manuales.

En el CDS-UTEZ, el Equipo de Trabajo esta integrado de la siguiente forma: 1 Responsable de Administración del Proyecto, 1 Responsable de Desarrollo y Mantenimiento de Software, 5 Analistas, que al mismo tiempo tienen la función de Programadores, 1 Diseñador de Interfaz de Usuario, 1 Diseñador. De este Equipo de Trabajo nos enfocaremos con el rol de los programadores, ya que es en esta etapa donde se desarrollan los componentes y donde deben realizarse las pruebas unitarias.

Se realiza una investigación de campo entrevistando al 100% de los programadores responsables de la generación de los componentes, en donde se detecta:

1. Que no se cuenta con una metodogía documentada y homogénea entre los programadores para la verificación de las pruebas unitarias.

2. El 100% realiza pruebas de funcionamiento, cumpliendo con especificaciones de requerimientos.

3. Que el 100% de los programadores no llevan a cabo las pruebas unitarias utilizando algún framework para pruebas unitarias.

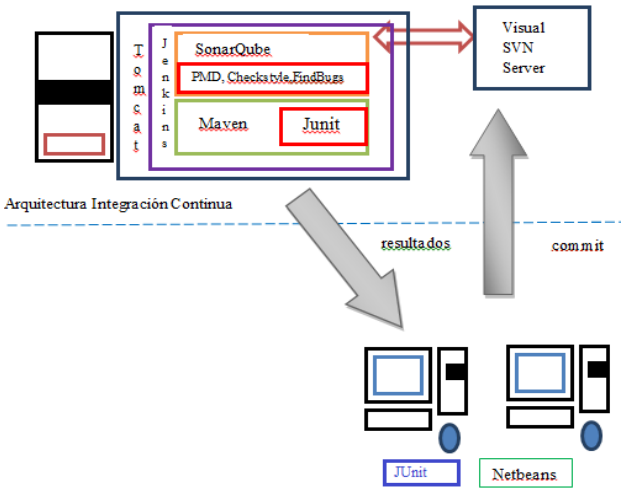
El proyecto se tomó de base para ser desarrollado utilizando la integración continua y aplicando las pruebas unitarias correspondientes. A continuación se presentan las herramientas de software que conforman la arquitectura de Integración Continua.

Tabla 10.1 Herramientas de la arquitectura de integración continúa para el sistema de encuestas

Herramienta de software	Tipo de herramienta	Versión utilizada
Netbeans	Entorno de desarrollo integrado	7.4
Jenkins	Servidor de Integración Continua	1.5
Visual SVN Server	Controlador de versiones	2.7.3
Apache Tomcat	Contenedor de Servlets	7.0
Apache Maven	Gestionador de proyectos	3.1
JUnit	Framework pruebas unitarias funcionales	1.4
SonarQube	Analizador de código estatico	3.7

En la figura se ve el diagrama de cómo quedó conformada la prueba para la integración continua del sistema de encuestas.

Figura 10.2 Herramientas de la arquitectura de integración continúa para el Sistema Integral de Servicios Académicos



10.1 Resultados

El proyecto se ha analizado tres veces durante este periodo de desarrollo, en las cuales se han aplicado pruebas unitarias con el fin de asegurar que cada componente que se va realizando, cumple con los requerimientos funcionales y que no presente en su estructura un posible error o algo que pueda generarlo.

En el primer análisis realizado no se encontraron errores en las pruebas funcionales ni estructurales, esto permitio seguir con el desarrollo, asegurando que los componentes probados funcionarian en la siguiente etapa de integración. Sin embargo, los resultados de cobertura mostraron que no todos los métodos contaban con una prueba unitaria. En las figuras 3 y 4 se observan los resultados.

Es importante conocer la cobertura de nuestro código debido a que es una métrica para saber cuál es el porcentaje de nuestro proyecto que está sometido a una prueba unitaria. Entre mayor cobertura, mayor parte de nuestro código cuenta con pruebas unitarias.

Figura 10.3 Resultado del primer análisis de pruebas unitarias

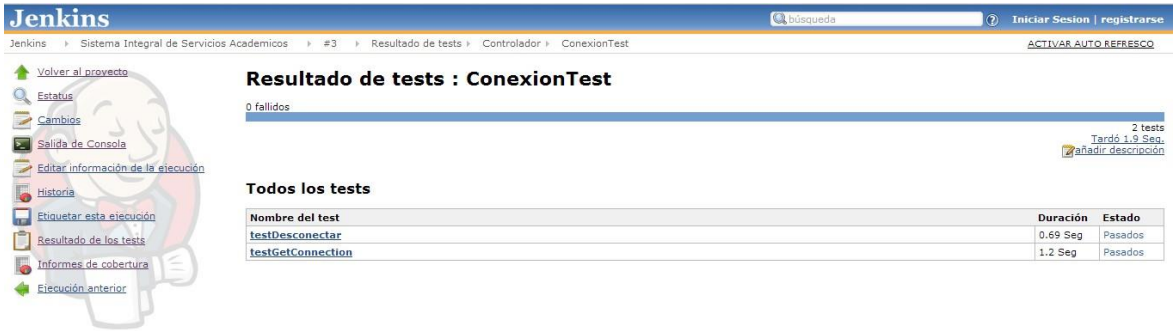
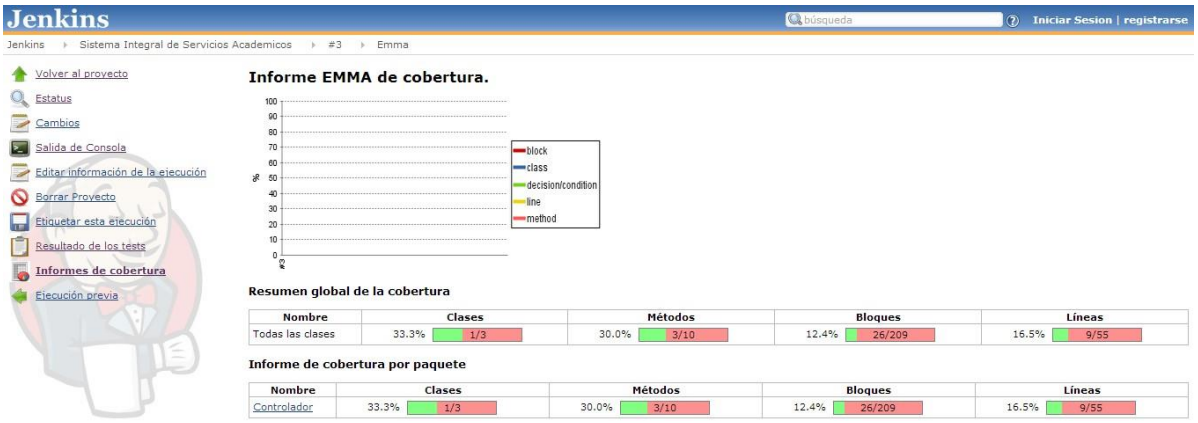


Figura 10.4 Resultados de cobertura

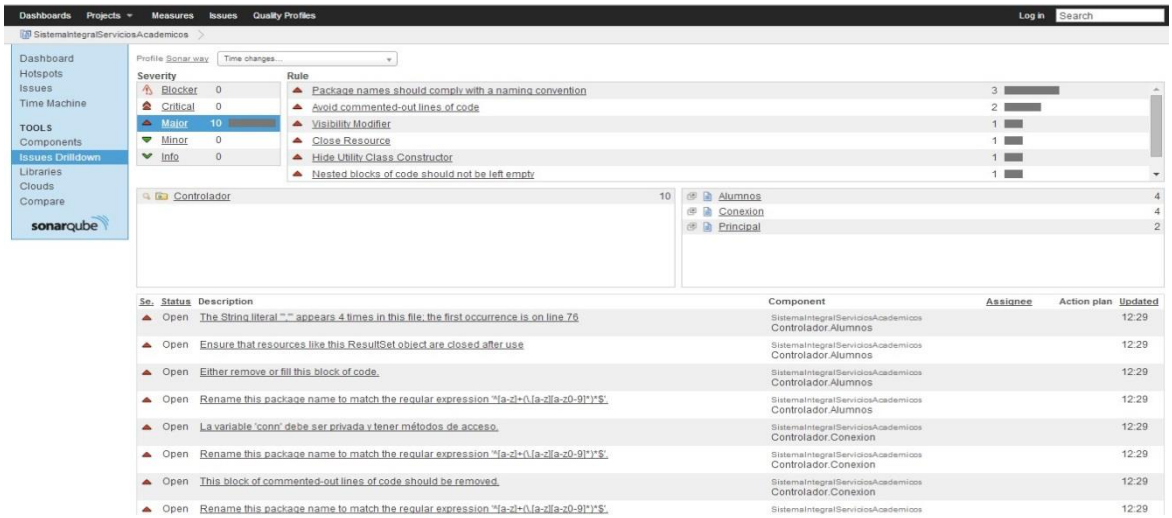


En el segundo análisis realizado se encontró que el 25% de los métodos elaborados presentaban un error funcional. En las pruebas estructurales no se encontraron errores, únicamente sugerencias hacia buenas prácticas en convenciones Java. En la figura 5 y 6 se observan los resultados de las pruebas.

Figura 10.5 Resultados del segundo análisis de pruebas unitarias

El error se encontró al momento de insertar en la base de datos, el cual fue corregido cuando se notifico el fallo, para asegurar que los demás componentes no salieran afectados en la etapa de integración.

Figura 10.6 Resultados del análisis estructural de SonarQube



El tercer análisis mostró que los errores se habían corregido, además de que la cobertura había aumentado. Los resultado del tercer análisis se observan en la figura.

Las pruebas deben realizarse lo mas continuamente posible, esto con la finalidad de que los componentes queden probados y se facilite la etapa de integración.

Figura 10.7 Resultados del tercer análisis

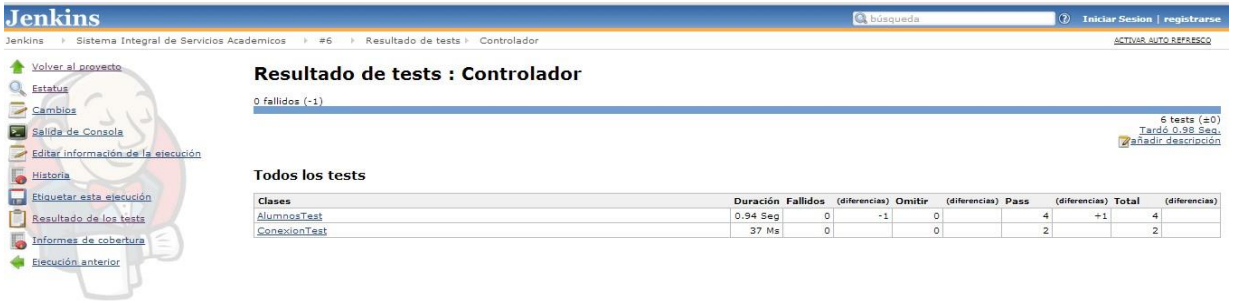
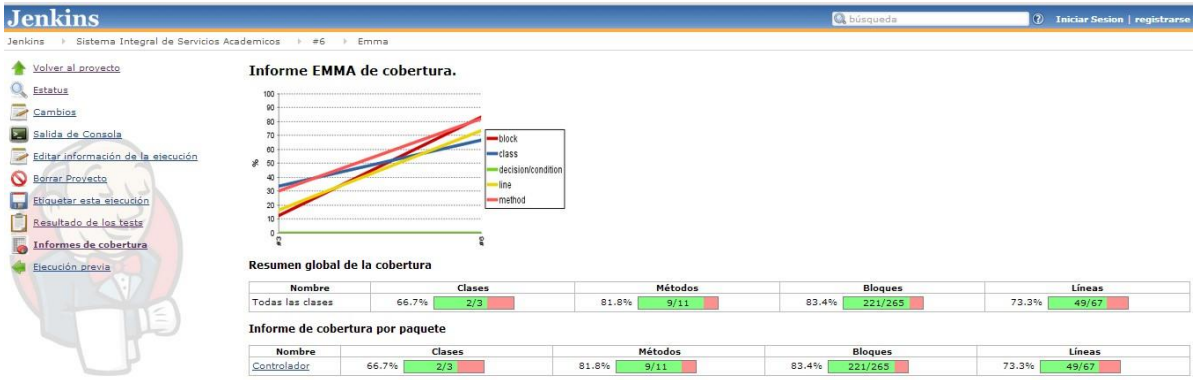


Figura 10.8 Resultados de cobertura



10.2 Conclusiones

Analizados los resultados obtenidos se concluye y se propone la implementación de herramientas y modelos que permitan mejorar la aplicación de pruebas unitarias dentro del proceso de Desarrollo y Mantenimiento de Software de la norma Moprosoft, con el fin de mejorar y guiar a las empresas hacia una mejora de calidad de software.

Referencias

Boletin UNAM- DGCS-528 [Online http://www.dgcs.unam.mx/boletin/bdboletin/2011_528.html]

UTEZ [Online http://www.utez.edu.mx/index.php/noticias-blog/234-la-utez-obtiene-el-nivel-3-de-maduracion-en-moprosoft]

NYCE [Online http://www.nyce.org.mx/index.php/proceso-verif/moprosoft]

NYCE [Online http://www.moprosoft.com.mx/contenido.aspx?id_pagina=1]

Oktaba H. (2003). Modelo de procesos para la industria del software, MoProSoft. Versión 1.1. Recuperado de [Online http://www.uv.mx/rrojano/MIS/desrrollo1/material/moprosoft-v1.1.pdf]

Osherove R. (2009). *The Art of Unit Testing: with Examples in .NET*. EUA. Manning Publications.

JUnit [Online http://junit.org/]

PMD [Online http://pmd.sourceforge.net/]

Checkstyle [Online http://checkstyle.sourceforge.net/]

FindBugs [Online http://findbugs.sourceforge.net/]

SonarQube [Online http://www.sonarqube.org/]

SonarQube Documentation [Online http://docs.codehaus.org/]

Tahchiev, P. & Leme, F. & Massol, V & Gregory, G. (2010). *JUnit in Action*. EUA. Manning Publications.