Universidad de las Ciencias Informáticas

Desarrollo de pruebas unitarias automatizadas a los módulos de seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

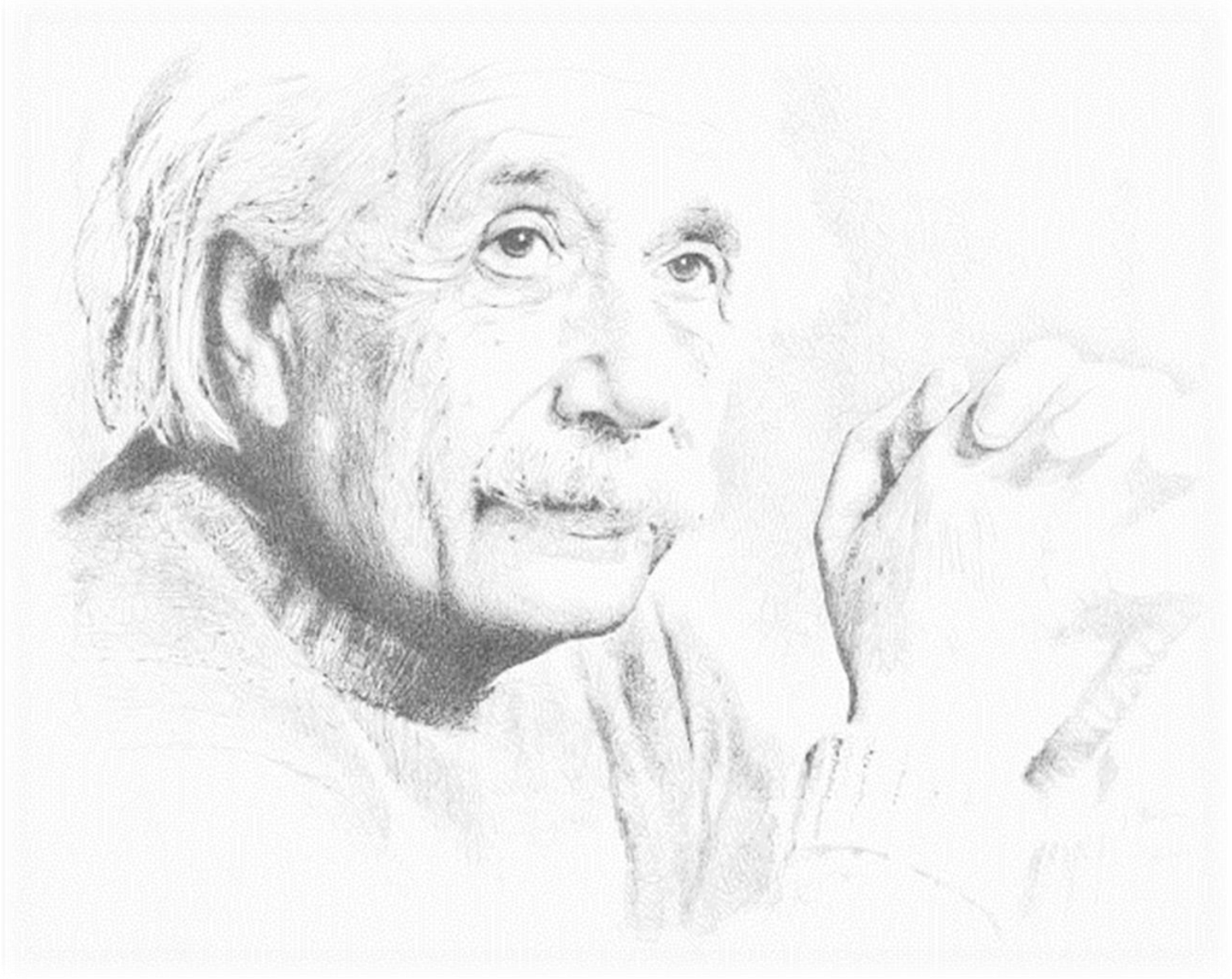
Autores:

Dairon Fermin González Ramirez
Asiel Ramírez Faez

Tutor:
Ing. Yampier Medina Tarancón

Co-Tutor:
M.Sc. Vladimir Martell Fernández

La Habana, junio del 2015
"Año 57 de la Revolución"



No pretendamos que las cosas cambien si siempre hacemos lo mismo.

Albert Einstein.

/

Declaramos ser los únicos autores del trabajo titulado: Desarrollo de pruebas unitarias automatizadas a los módulos de seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0.

Se cede al Centro de Geoinformática y Señales Digitales de la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmamos la presente a los 23 días del mes de junio del año 2015.

Dairon F. González Ramirez	Asiel Ramírez Faez
Autor	**Autor**
Ing. Yampier Medina Tarancón	M. Sc. Vladimir Martell Fernández
Tutor	**Co-Tutor**

Síntesis del Tutor:

Ing. Yampier Medina Tarancón

Ingeniero en Ciencias Informáticas graduado en el 2009 en la Universidad de las Ciencias Informáticas. Ha desempeñado los roles de planificador, asesor de calidad, jefe de grupo de calidad, probador y revisor. Diplomado en Docencia e Innovación Universitaria. Profesor Asistente. Impartió las asignaturas curriculares de Historia de la Informática, Gestión de Software, y Componente Profesional de Ingeniería y Gestión de Software. Además de las asignaturas optativas de Introducción a la Geoinformática e Infraestructura de Datos Espaciales.

E-mail: ytarancon@uci.cu

Síntesis del Co-Tutor:

M.Sc. Vladimir Martell Fernández.

Graduado de Ingeniería en Ciencias Informáticas en la Universidad de las Ciencias Informáticas en el año 2008. Es profesor con categoría de Asistente, de la facultad 6. Actualmente imparte la asignatura de Programación 1. Pertenece al Centro de Geoinformática y Señales Digitales en el cual se desempeña como Jefe del Grupo de Inteligencia Empresarial.

E-mail: vmartell@uci.cu

De Dairon:

Primeramente quisiera agradecer a cada persona tanto profesor como no docente que ha contribuido de una forma u otra a mi formación como ingeniero, formación que comenzó desde la enseñanza primaria hasta hoy día. A mis tutores Yampier Medina Tarancón y Vladimir Martell Fernández, por apoyarme, guiarme y ayudarme a conseguir este difícil objetivo, el mérito es tanto mío como de ellos.

A mis amigos sin que me quede ninguno, pero en especial a esta hermosa familia que hemos formado en la UCI desde primer año y que luego creció en tercero, sin más, al grupo 6503. En especial a Leo Danny, Luis Alberto, Andrés y Sergio por estar en fiestas, pruebas, comidas, en fin, para lo que se necesite en estos difíciles cinco años. También a mi dúo de tesis por su apoyo durante este difícil proceso.

A mis suegros por estar siempre preocupados, presentes en todo tipo de momentos, brindando su apoyo y consejos a lo largo de estos tres años. A mi novia Claudia por brindarme siempre su comprensión, ayuda, cariño y como no, sus regaños. Por ser esa persona en la que podía confiar sin importar el momento, ni las circunstancias.

A toda mi familia por confiar siempre en que podía alcanzar mis metas y darme fuerzas para seguir adelante. En especial a mi tío, que más que tío es mi hermano Adalberto, por ver siempre en mi ese hermanito que puede lograr todo lo que se proponga, por su apoyo, sus consejos y preocupaciones. A Ramón por ser siempre un apoyo incondicional tanto en las buenas como en las malas. A mi abuela Berta por ser mi segunda mamá, por brindarme su cariño incondicional y preguntarme cada día durante estos cinco años cuando iba a la casa.

A mis padres por ser un ejemplo, por comprenderme, apoyarme en cada decisión. A pipo por brindarme todo lo que tiene, por apoyarme en cuanta cosa se me ocurre, por mostrarme su orgullo y confiar siempre en que puedo lograr lo que me preponga aunque ni yo mismo lo crea posible. Para mi mamá no tengo palabras, ni me va a alcanzar el tiempo, ni la vida para agradecerle todo lo que ha hecho, sigue y seguirá haciendo por mí, solo tengo dos palabras para ella, gracias a ti.

De Asiel:

Primeramente quisiera agradecer a cada uno de los seres que han intervenido en este logro, especialmente quiero a agradecer a mis padres y familiares por creer en que llegar a esta instancia de la vida se podía, estando siempre ahí dándome los mejores consejos y haciéndome las mejores críticas constructivas, siempre para hacer de mí una mejor persona.

Agradecer también a los señores Dairon Fermin González Ramirez, Yampier Medina Tarancón, y Vladimir Martell Fernández por haber sido motores impulsores para el buen desarrollo de esta investigación para optar por tan preciado título.

Agradecimiento especial a todos los personajes aquí en la universidad que a lo largo de mi trayectoria estuvieron siempre dándome apoyo en todo momento. A todos gracias de corazón.

De Asiel:

Quiero regalar y hacer partícipe de este triunfo a:

Mi abuela: Por haber sido y ser hoy más que nunca tan especial en mi vida y sentirse desde donde está muy orgullosa de su nieto.
A mis padres que sin lugar a duda nunca dejaron de ser junto a abuelos, tíos, tía, primos y hermana, fundamentales para que yo pudiese llegar a la meta.

De Dairon:

Quiero dedicar el fruto de tantos años de esfuerzo y dedicación a:

Mis abuelas: Por ser especiales y confiar siempre en mí a quienes les dedo también parte de lo que soy.
Mis padres: Por ser maravillosos, pacientes, regalarme mi vida y la de ellos también. Por ello quiero darles este pequeño regalo.

Índice de contenido

Índice de Figuras

Índice de Tablas

Resumen

La Universidad de las Ciencias Informáticas es un centro docente-productor que desarrolla aplicaciones y servicios informáticos orientados a diversos sectores de la economía, dentro y fuera de Cuba, a partir de la vinculación estudio-trabajo como modelo de formación. XILEMA PRIMICIA es un sistema audiovisual que se desarrolla en el Centro de Geoinformática y Señales Digitales de esta Universidad y fue concebido para divulgar información a través de una red televisiva interna o vía satélite. Aún cuando el proceso de pruebas de software incide directamente en la calidad de XILEMA PRIMICIA, este presenta importantes limitaciones en el ciclo de vida del desarrollo de cada versión: Detección de no conformidades en etapas tardías, disminución del tiempo de ejecución del proceso de pruebas en función del cumplimiento de los cronogramas de entrega pactados, imposibilidad de desarrollar algunos tipos de pruebas por la complejidad de su ejecución manual, entre otros.

La presente investigación tiene como objetivo desarrollar una colección de pruebas unitarias automatizadas[1] a los módulos de seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0 para favorecer la productividad y garantizar los niveles de calidad que se esperan. Se desarrollan todas las fases del proceso de pruebas unitarias utilizando el framework para pruebas unitarias PHPUnit y finalmente se establecen un conjunto de análisis con datos relevantes para la calidad de software que validan la puesta en práctica del proceso de automatización de pruebas unitarias realizado.

Palabras clave: Calidad; PRIMICIA; pruebas automatizadas; pruebas unitarias.

[1] Conjunto de pruebas unitarias almacenadas previamente diseñadas e implementadas.

Introducción

El auge mundial alcanzado por las Tecnologías de la Información y las Comunicaciones ha propiciado el considerable avance del quehacer científico, experimentando este último un crecimiento notablemente alto. La innovación tecnológica ha incrementado de manera significativa la producción de aplicaciones informáticas, permitiendo que el tamaño y la complejidad del software aumenten rápidamente, al mismo tiempo en que sus clientes y usuarios son cada vez más exigentes; alcanzar productos de alta calidad se ha convertido en un desafío para los desarrolladores de software.

La Universidad de las Ciencias Informáticas (UCI) es un centro docente-productor que desarrolla aplicaciones y servicios informáticos orientados a diversos sectores de la economía, dentro y fuera de Cuba, a partir de la vinculación estudio-trabajo como modelo de formación. Con notable experiencia en su quehacer, la cartera comercial de la UCI se orienta a cinco áreas temáticas de alto impacto, con productos de software libre como protagonistas; cuya calidad avalan clientes nacionales y extranjeros (UCI, 2012).

XILEMA PRIMICIA es un sistema audiovisual que se desarrolla en el Centro de Geoinformática y Señales Digitales de la UCI, fue concebido para divulgar información a través de una red televisiva interna o vía satélite. Tiene como objetivo fundamental proveer un canal de televisión, donde se transmitan las informaciones que considere de importancia el usuario. PRIMICIA se encuentra dividido en dos subsistemas: Transmisión y Administración. El subsistema de Transmisión se encarga de la trasmisión de las informaciones en forma de noticias e infocintas que son creadas con anterioridad desde el subsistema de Administración, haciendo uso de los recursos multimedia que se encuentran en el sistema. Este subsistema está compuesto por seis módulos, dentro de los cuales se encuentran seguridad y gestión de medias. (PRIMICIA, 2013).

El proceso de pruebas de software incide directamente en la calidad del producto final obtenido, enunciadas formalmente, según (Jovanović, 2006), las pruebas de software son una actividad realizada para evaluar la calidad del software y mejorarla. Por lo tanto, el objetivo es la detección sistemática de diferentes clases de errores en una cantidad mínima de tiempo y con una cantidad mínima de esfuerzo.

De acuerdo con un artículo publicado por (IEEE, 2014) la planificación de las pruebas de software debe comenzar con las primeras etapas del proceso de desarrollo de software, […] deben ser sistemáticamente desarrolladas y continuamente refinadas a medida que avanza el desarrollo de software.

No obstante lo anterior, el proceso de pruebas en el ciclo de vida del desarrollo de cada versión de PRIMICIA presenta importantes limitaciones como se expresa a continuación:

- Detección de no conformidades en etapas tardías, durante la integración de los subsistemas, resultando más costosa su corrección.
- Variación más acentuada del tiempo de desarrollo con respecto a la planificación, determinada por la imposibilidad de predecir o planificar la resolución de no conformidades complejas en etapas donde el costo es muy alto.
- Disminución del tiempo de ejecución del proceso de pruebas en función del cumplimiento de los cronogramas de entrega pactados, comprometiendo la calidad del producto obtenido.
- Imposibilidad de desarrollar algunos tipos de pruebas por la complejidad de su ejecución manual.

De acuerdo con la problemática presentada se propone el siguiente **problema de investigación:** ¿Cómo disminuir las limitaciones del proceso de pruebas en el ciclo de vida del desarrollo de PRIMICIA 2.0 que proporcione mayor productividad y garantice los niveles de calidad esperados del producto obtenido?

Para dar solución al problema de investigación se propone como **objetivo general:** desarrollar una colección de pruebas unitarias automatizadas a los módulos de seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0 para favorecer la productividad y garantizar los niveles de calidad esperados.

Se define como **objeto de estudio:** el proceso de pruebas en el desarrollo del software, enmarcando el **campo de acción** en**:** la automatización del proceso de pruebas unitarias en los módulos seguridad y gestión de medias del sistema XILEMA PRIMICIA.

Se elaboran las siguientes **preguntas científicas** que sirven de guía para la investigación:

1. ¿Cuáles son las dificultades que existen el proceso de pruebas?
2. ¿Qué proceso debe ser aplicado para mejorar la productividad en las pruebas de software?
3. ¿Qué herramientas y tecnologías deben ser empleadas en el desarrollo de las pruebas unitarias?

4. ¿Qué características debe tener la colección de pruebas para garantizar la reducción de errores?
5. ¿Cómo determinar que la colección automatizada de pruebas unitarias favorecerá la productividad durante las pruebas de software?

Para dar cumplimiento al objetivo general trazado se definen las siguientes **tareas de investigación:**

1. Caracterizar las pruebas en el desarrollo de software.
2. Caracterizar las pruebas unitarias.
3. Seleccionar las tecnologías y las herramientas a utilizar durante el desarrollo de las pruebas unitarias.
4. Diseñar un conjunto de casos de prueba de acuerdo a las funcionalidades de los módulos de seguridad y gestión de medias.
5. Implementar las pruebas unitarias de acuerdo a la metodología seleccionada.
6. Realizar el análisis de los resultados obtenidos.

La investigación se desarrolla utilizando los **métodos científicos** teóricos y empíricos, específicamente los que se argumentan a continuación:

Métodos teóricos:

Histórico - Lógico: Utilizado para concretar teóricamente la evolución del flujo de pruebas, en el proceso de desarrollo de software haciendo énfasis en las pruebas unitarias, identificando de estas últimas tendencias, prácticas, estrategias y tecnologías asociadas.

Analítico - Sintético: Utilizado para extraer los elementos y flujos de información más importantes de la bibliografía sobre las pruebas unitarias. También sintetizar la tecnología y el entorno con que fue desarrollado el sistema XILEMA PRIMICIA 2.0.

Métodos empíricos:

Análisis documental: Utilizado para el estudio del amplio dominio de la bibliografía consultada y resaltar los aspectos más importantes de la misma, apreciando información más certera sobre el tema.

El contenido de la presente investigación se encuentra estructurado en 3 capítulos, organizados de la siguiente manera:

Capítulo 1: Fundamentación teórica

Exposición de los principales conceptos, tendencias actuales e ideas asociadas al proceso de pruebas como parte del desarrollo de software, haciendo énfasis en la automatización de las pruebas unitarias. Además, se proponen las tecnologías y herramientas necesarias para la construcción de la propuesta.

Capítulo 2: Planificación, diseño, desarrollo

Aborda la aplicación de las diferentes fases del proceso de pruebas unitarias identificado, además de la utilización del framework para pruebas unitarias PHPUnit 4.5.0, formando así parte fundamental del proceso ingenieril.

Capítulo 3: Evaluación

Contiene parte fundamental de la investigación, establece un conjunto de análisis con datos relevantes para la calidad de software que validan la puesta en práctica del proceso de automatización de pruebas unitarias realizado.

Capítulo 1: Fundamentación teórica

1.1 Introducción

El presente capítulo expone el fundamento teórico relacionado con el problema de la investigación. Para dar cumplimiento al objetivo propuesto se hace un estudio de los conceptos y tendencias fundamentales asociados a la investigación. Finalmente, se realiza un análisis de las herramientas y tecnologías a utilizar en el desarrollo de la solución.

1.2 Calidad de Software

Actualmente en el mercado de software existen índices de competitividad extremadamente altos. Añadiendo a esto la incansable búsqueda por parte de los clientes de productos que satisfagan sus necesidades se hace inevitable realizar sistemas computacionales con calidad. Por lo tanto es inminente preguntarse ¿Qué es calidad de software?

La concepción de calidad por algunas instituciones y organismos se puede comparar con los copos de nieve, porque a simple vista parecen iguales, pero en realidad todos son distintos de ahí que cada entidad utiliza su propia versión, dentro de las que se encuentran algunas bien difundidas como en los siguientes casos:

- Grado en el que un conjunto de características inherentes cumple con los requisitos (ISO, 2000).
- La concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo software desarrollado profesionalmente (Pressman, 2002).
- El grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario (IEEE, 1990).

Simplificando y uniendo los diferentes puntos conceptuales se define calidad de software como características concordantes con los requisitos especificados, acorde con su utilidad y existencia, persiguiendo siempre la conformidad del cliente o consumidor.

1.2.1 Gestión de la calidad

La calidad de software no constituye un producto sino un proceso. Siguiendo la denominación del Instituto de Administración de Proyectos (del inglés, Project Management Institute, PMI), la Gestión de la calidad del proyecto incluye los procesos y actividades de la organización ejecutante que determinan responsabilidades, objetivos y políticas de calidad a fin de que el proyecto satisfaga las necesidades por la cuales fue emprendido (PMBOOK, 2009).

Los procesos que componen la Gestión de la calidad del proyecto se denominan:

- **Planificación de la calidad:** La planificación de cualquier etapa del proceso de desarrollo de software es de vital importancia ya que constituye la relación base entre los objetivos y recursos de la organización. Es el proceso por el cual se identifican los requisitos de calidad y/o normas para el proyecto y el producto, documentando la manera en que el proyecto demostrará el cumplimiento con los mismos (PMBOOK, 2009).
- **Aseguramiento de la calidad (SQA):** El SQA procura mantener la calidad a lo largo de todo el desarrollo y mantenimiento del producto mediante la ejecución de una variedad de actividades en cada etapa que pueden permitir identificación temprana de problemas, un rasgo casi inevitable de cualquier actividad compleja (IEEE, 2004). La garantía de calidad consiste en la auditoría y las funciones de información de la gestión (Pressman, 2002).
- **Control de la calidad:** Proceso por el cual se monitorizan y registran los resultados de la ejecución de actividades de control de calidad, a fin de evaluar el desempeño y recomendar cambios necesarios (PMBOOK, 2009).

La gestión total de la calidad y las filosofías similares fomentan una mejora continua de procesos, que conduce al desarrollo de enfoques cada vez más robustos para la ingeniería del software (Letelier, 2006).

1.3 Productividad

La productividad es un fenómeno que está estrechamente relacionado con la calidad. A continuación se enuncian un conjunto de criterios de los más difundidos para definir la productividad:

- Factor que permitirá medir cuánta es la brecha entre el potencial de valor que pueden generar los recursos y lo que actualmente están generando, de manera que el personal esté pensando en la innovación constante y por ende que la organización sea trascendental en el tiempo (Castañeda Vargas y Guevara Sandoval, 2010).
- La productividad es definida como el resultado de la articulación armónica entre la tecnología, la organización y el talento humano, combinando en forma óptima o equilibrada los recursos para la obtención de los objetivos (Cequea y Rodríguez Monroy, 2012).
- La productividad es una proporción entre las entradas y salidas (Hernández Meraz, 2009).

Se puede sintetizar que el termino productividad se refiere a la relación o correspondencia entre los recursos del proyecto y la calidad de sus productos. Por tanto se hace necesario establecer una relación concreta entre la calidad y la productividad.

1.4 Pruebas de software

En los últimos años, la visión de las pruebas de software ha madurado hasta transformarse en un proceso iterativo e incremental. Las pruebas ya no son vistas como una actividad que se inicia sólo después de la fase de codificación sino a lo largo del proceso de desarrollo de software.

La planificación de las pruebas de software debe comenzar con las primeras etapas del proceso de desarrollo de software, los planes de pruebas y procedimientos deben ser sistemáticamente desarrollados y continuamente refinados a medida que avanza el desarrollo de software (IEEE, 2014).

Por lo tanto se hace indispensable conocer en qué consisten las pruebas de software. A continuación se reflejarán concepciones difundidas:

- Las pruebas de software consisten en la verificación dinámica de los comportamientos esperados que ofrece un programa en un conjunto finito de casos de prueba, seleccionados adecuadamente del dominio de ejecución generalmente infinito (IEEE, 2014).
- Las pruebas son una actividad realizada para evaluar la calidad del software y mejorarla. Por lo tanto, el objetivo es la detección sistemática de diferentes clases de errores en una cantidad mínima de tiempo y esfuerzo (Jovanović, 2006).

- Las pruebas se definen como una actividad en la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan, se registran y se realiza una evaluación de algún aspecto (Fernández Sanz, 2005).

1.4.1 Objetivos de las pruebas de software

Las pruebas pueden ser dirigidas a la verificación de diferentes propiedades. Los casos de prueba pueden ser diseñados para comprobar que las especificaciones funcionales se ejecutan correctamente, se hace referencia en la literatura como pruebas de conformidad, las pruebas de corrección, o las pruebas funcionales. Sin embargo, otras propiedades no funcionales pueden ser probadas (IEEE, 2014).

A continuación se definen una serie de objetivos estandarizados para las pruebas (Pressman, 2002):

- La prueba es el proceso de ejecución de un programa con la intención de descubrir un error.
- Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces.
- Una prueba tiene éxito si descubre un error no detectado hasta entonces.

1.4.2 Características de las pruebas de software

Es de importancia significativa conocer los atributos o características básicas de las pruebas, para reconocer cuándo se está en presencia de una buena prueba.

- Una buena prueba tiene una alta probabilidad de encontrar un error.
- Una buena prueba no debe ser redundante. El tiempo y los recursos para las pruebas son limitados. No hay motivo para realizar una prueba que tiene el mismo propósito que otra. Todas las pruebas deberían tener un propósito diferente (incluso si es sutilmente diferente).
- Una buena prueba no debería ser ni demasiado sencilla, ni demasiado compleja. Aunque es posible a veces combinar una serie de pruebas en un caso de prueba, los posibles efectos secundarios de este enfoque pueden enmascarar errores. En general, cada prueba debería realizarse separadamente.

1.4.3 Tipos de pruebas de software

Unitarias: Las pruebas unitarias verifican el funcionamiento de forma aislada de elementos de software que son comprobables por separado. Normalmente se prueban los métodos de las clases eliminando las dependencias que existan con otras estructuras lógicas. Típicamente, las pruebas de unidad se producen con acceso al código a prueba (IEEE, 2014).

Integración: Las pruebas de integración son una técnica sistemática para construir la estructura del programa mientras que, al mismo tiempo, se llevan a cabo pruebas para detectar errores asociados con la interacción de los componentes de software (Pressman, 2002).

Funcionales: Las pruebas funcionales validan partes de las aplicaciones informáticas y simulan la navegación del usuario, realizan peticiones y comprueban la respuesta para validar que una determinada acción se corresponde con los requerimientos especificados (Potencier y Zaninotto, 2012).

Sistema: Las pruebas del sistema se ocupan de probar el comportamiento de un sistema entero. Por lo general se considera adecuado para la evaluación de los requisitos no funcionales del sistema, tales como la seguridad, velocidad, precisión y fiabilidad (IEEE, 2014)

Aceptación: Las pruebas de aceptación determinan si un sistema satisface los criterios de aceptación, por lo general se chequean los comportamientos deseados del sistema contra los requerimientos del cliente. El cliente o representativos del mismo se compromete directamente en las actividades para comprobar que se han cumplido sus requisitos (IEEE, 2014).

1.5 Las pruebas unitarias automatizadas

La mayoría de los proyectos de las empresas productoras de software entran en conflictos cuanto las variables de tiempo y esfuerzo atentan contra el proceso de pruebas. En función de no incumplir los plazos, aún cuando existen desajustes en el cronograma, se opta por realizar menos pruebas. Mientras menos pruebas se realizan, menos eficaz es el código y el producto se convierte en inestable. Es por eso que las organizaciones entregan productos a sus clientes con un 15% de los defectos aún presentes en los productos (INTI Argentina, 2005).

Probar el software gradualmente en los módulos, es una de las tareas más importantes y básicas que se realiza en los procesos de desarrollo de software. La prueba de unidad centra el proceso de verificación en la menor unidad del diseño del software: el componente software o módulo (Pressman, 2002). Las pruebas que se obtienen como parte de las pruebas de unidad están esquemáticamente ilustradas en la Figura 1.

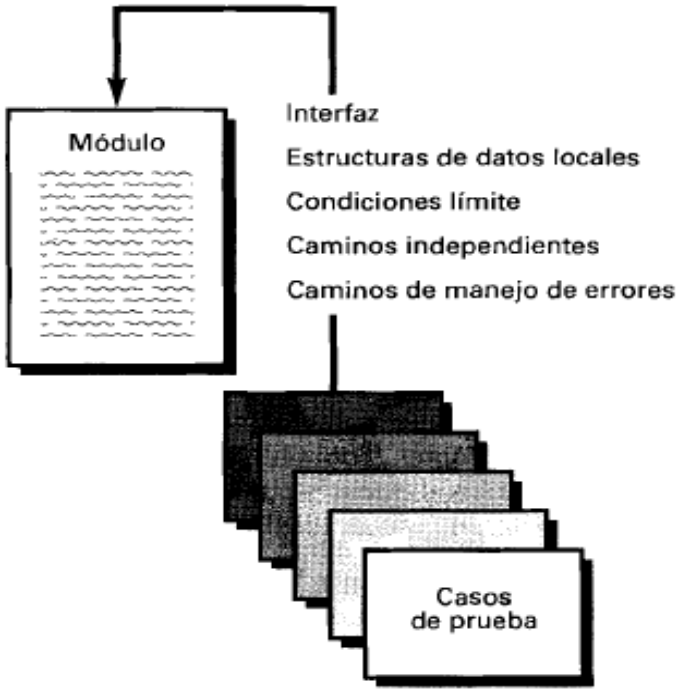


Figura 1: Pruebas de unidad (Pressman, 2002).

Las pruebas de unidad son realizadas por los desarrolladores respectivos en las unidades individuales de código fuente de las áreas que le fueron asignadas. La meta de pruebas de unidad es aislar cada parte del programa y mostrar que las partes individuales son correctas por lo que se refiere a los requisitos y funcionalidades (Simply Easy Learning, 2010). Estas pruebas constituyen la técnica de verificación y validación, donde el programador gana la seguridad de las unidades individuales del código fuente (Khaled, Rafa y Mohammad, 2009).

Según la (IEEE, 1993), la prueba de unidad se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño del software (módulo). Una tarea esencial es la prueba selectiva de las rutas de

ejecución, con el fin de descubrir errores debidos a los cálculos incorrectos, comparaciones erróneas o flujos de control inapropiados.

Por su parte, (IBM, 2006) considera que la prueba de unidad se centra en la verificación de los elementos más pequeños del software que se puedan probar. Normalmente, las pruebas de unidad se aplican a componentes representados en el modelo de implementación, para verificar que se cubren los flujos de control, los flujos de datos y que funcionan como se esperaba. El desarrollador realiza la prueba de unidad mientras se desarrolla la unidad. Los detalles de la prueba de unidad se describen en la disciplina de implementación

Los autores entienden que las pruebas unitarias son la verificación realizada a la unidad menor del código de una aplicación informática (métodos que componen las clases), en aras de detectar errores debidos a los cálculos incorrectos, comparaciones erróneas o flujos de control inapropiados mediante las distintas rutas de ejecución a edades tempranas del proceso productivo.

1.5.1 Identificación y ubicación de la etapa de pruebas unitarias

El proceso de pruebas de un sistema computacional no termina con las pruebas de unidad sino que estas son prácticamente el comienzo del necesario y valioso proceso. Comienza probando el software gradualmente en los módulos hasta que estos estén completados (Pruebas de unidad); entonces se prueban los grupos de módulos probados integrados con los módulos recientemente completados (Prueba de integración). Este proceso continúa hasta que todos los módulos del paquete se hayan probado. Una vez esta fase se completa, el paquete entero se prueba en conjunto (Prueba del sistema). Esta estrategia de la comprobación normalmente es el término conocido como comprobación incremental (Konka, 2011).

A continuación se muestra la Figura 2 que se corresponde con el proceso incremental al que se hace alusión en la idea anterior.

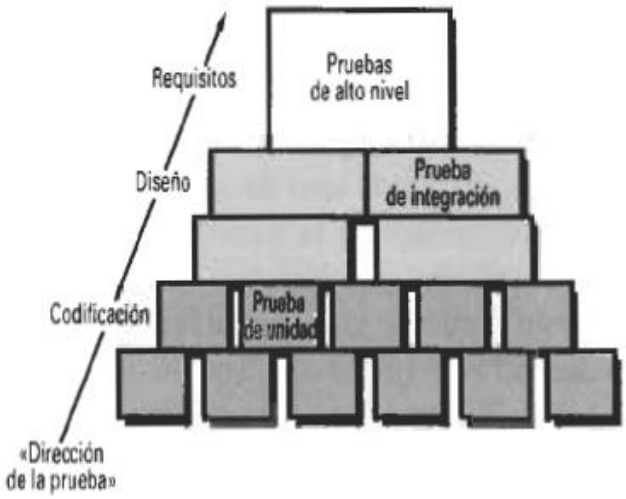


Figura 2: Etapas en la prueba del software (Pressman, 2002).

1.5.2 Características de las pruebas unitarias

Es necesario tener en cuenta una serie de características o requisitos para que las pruebas unitarias cumplan su objetivo o finalidad (Joyanes Aguilar, 2007):

- Automatizable: No debería requerirse una intervención manual. Esto es especialmente útil para la integración continua.
- No requiere de mucho código para prepararla: Una prueba de unidad cuya preparación toma 3 minutos (iniciando el ambiente y las conexiones) se debe considerar demasiado pesada para ejecutarse cada vez que se hagan cambios.
- Repetibles o Reutilizables: No se deben crear pruebas que sólo puedan ser ejecutadas una sola vez.
- Independientes: La ejecución de una prueba no debe afectar a la ejecución de otra.
- Probar sólo un método: Aunque no siempre es necesario hacer "una prueba por método", sí es importante asegurar que cubre el código. Muchas veces se requiere más de un método de prueba, por cada método que se dispone, para poder probar los datos de entrada y los posibles resultados.
- Profesionales: Las pruebas deben ser consideradas igual que el código, con la misma profesionalidad y documentación.
- Dejar los datos de la misma manera en que los encontró: Cuando la prueba comienza, los datos se preparan si es necesario, pero cuando la prueba termina, es importante que el sistema borre los datos después de la verificación.

1.5.3 Ventajas que proporcionan las pruebas unitarias

Las pruebas unitarias son de incalculable ayuda para los desarrolladores, contribuyen a lo largo del proceso de desarrollo de software a la calidad del producto. Estas pruebas aisladas proporcionan cinco ventajas básicas (Joyanes Aguilar, 2007):

- Fomentar el cambio: Las pruebas unitarias facilitan que el programador cambie el código para mejorar su estructura (refactorización), puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.
- Simplificar la integración: Puesto que permiten llegar a la fase de integración con un grado alto de seguridad que el código está funcionando correctamente.
- Documentar el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.
- Separación de la interfaz y la implementación.
- Los errores están más limitados y fáciles de localizar dado que las pruebas unitarias pueden desenmascararlos directamente en el método que presenta error, evitando de esta manera la propagación de los errores a través de dependencias.

1.6 Procedimiento para la automatización de pruebas unitarias

Para la puesta en práctica de un proceso de automatización de pruebas unitarias es imprescindible seguir el estándar para pruebas unitarias de software (IEEE Std. 1008-1987) actualizado en el año 1993, el mismo es recomendado en la Guía para el Conocimiento de Ingeniería de Software (SWEBOK).

Este estándar especifica el proceso de pruebas unitarias logrando un enfoque uniforme para las pruebas unitarias, además de describir los conceptos de ingeniería de software y pruebas en que basa su enfoque, lo cual proporciona orientación y recursos de información para ayudar con la implementación y el uso del proceso de pruebas unitarias (IEEE, 2014).

1.6.1 Actividades del proceso de pruebas unitarias

Las pruebas unitarias constituyen una de las actividades más importantes del proceso de desarrollo de software, a veces requiere más esfuerzo y tiempo que cualquier otra actividad de la ingeniería de software

(Pressman, 2002). Es importante, por tanto, establecer diferentes actividades que guíen el proceso de pruebas de unidad.

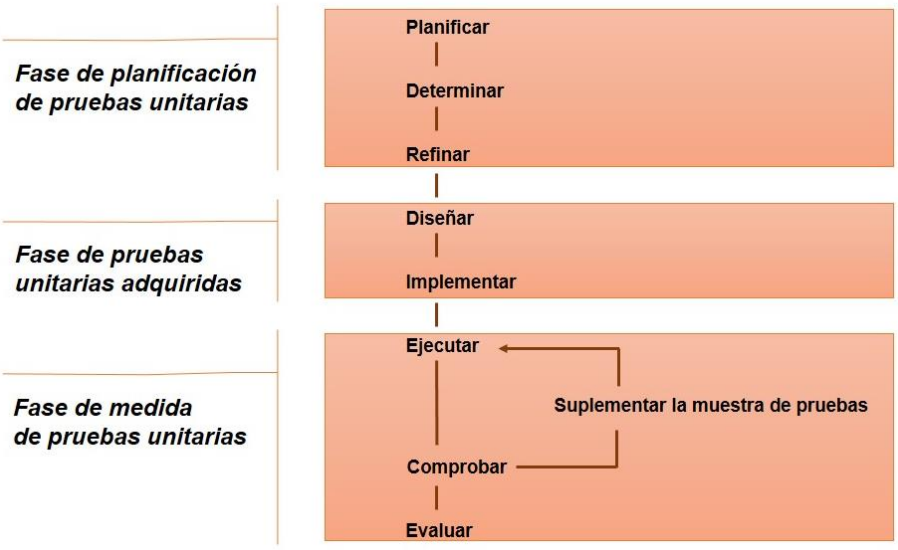


Figura 3: Flujo de actividades del proceso de pruebas unitarias (IEEE, 1993).

La Figura 3 muestra la secuencia de actividades involucradas en el proceso de pruebas unitarias enmarcadas en tres fases definidas.

A continuación se definen las tareas a realizar en cada una de estas actividades según (IEEE, 1993):

Planificar el enfoque general, recursos y calendario

Esta actividad inicia la primera fase del proceso de pruebas unitarias propuesto y está compuesta por las siguientes tareas:

- Especificar un enfoque general de las pruebas unitarias: Identificar áreas de riesgo que debe abordar la prueba, las fuentes de datos existentes, las técnicas generales para la validación de datos.
- Especificar restricciones sobre determinadas características: Describir provisiones para la aplicación de software que interactúa directamente con las unidades a ensayar.
- Especificar los requisitos de integridad: Identificar las áreas (características, procedimientos, estados, funciones, datos, instrucciones) para ser cubiertos por el conjunto de prueba de unidad y el grado de cobertura requerida para cada área.

- Especificar los requisitos de terminación: Especificar los requisitos para la terminación normal del proceso de pruebas unitarias. Identificar las condiciones que podrían causar la terminación anormal del proceso de pruebas de unidad y los procedimientos de notificación que se aplican.
- Determinar las necesidades de recursos: Estimar los recursos necesarios para la adquisición del equipo de prueba, ejecución inicial, y la posterior repetición de actividades de prueba.
- Especificar cuadro general: Horario limitado por los recursos y la disponibilidad de las pruebas unitarias para todas las actividades de pruebas unitarias.

Determinar características para ser probadas

- Estudiar los requisitos funcionales: Estudiar cada función descrita en la documentación de requisitos del módulo.
- Identificar requisitos adicionales y procedimientos asociados: Identificar requisitos distintos de funciones (rendimiento, atributos o restricciones de diseño) asociados con características de software que pueden ser probadas con eficacia a nivel de unidad.
- Identificar los estados de la unidad: Si la documentación de los requisitos de la unidad implica múltiples estados (inactiva, listo para recibir, procesar). Identificar cada estado y cada transición de estado válido.
- Identificar características de datos de entrada y salida: Identificar las estructuras de datos de entrada y de salida de la unidad que desea probar.

Refinar el plan general

- Acotar el enfoque: Identificar los casos de prueba existentes y procedimientos de prueba que deben ser considerados para su uso.
- Especificar las necesidades de recursos especiales: Identificar los recursos especiales que se necesitan para poner a prueba la unidad. Hacer los preparativos para los recursos identificados.

Diseñar el conjunto de pruebas

Esta actividad inicia la segunda fase del proceso de pruebas unitarias propuesto y está compuesta por las siguientes tareas:

- Diseñar la arquitectura de la prueba: Diseñar un conjunto jerárquicamente descompuesto de objetivos de la prueba de modo que cada objetivo de nivel más bajo se puede probar directamente por unos pocos casos de prueba.
- Obtener las especificaciones de casos de prueba: Especificar los nuevos casos de prueba.

Implementar el plan y el diseño refinado

- Obtener y verificar datos de prueba: Obtener copia de los datos de prueba existentes para ser modificados o utilizados sin modificación. Generar nuevos datos requeridos. Incluir datos adicionales necesarios para garantizar la coherencia y la integridad de los datos.
- Obtener recursos especiales: Obtener los recursos de apoyo de prueba especificados.
- Desarrollar los casos de pruebas: Implementar los casos de pruebas diseñados y especificados en la fase anterior.

Ejecutar los procedimientos de prueba

Esta actividad inicia la tercera fase del proceso de pruebas unitarias propuesto y está compuesta por las siguientes tareas:

- Ejecutar pruebas: Configurar el entorno de prueba. Ejecutar el conjunto de pruebas unitarias. Registrar todos los incidentes arrojados.
- Determinar los resultados: Para cada caso de prueba, determinar si la unidad pasa o no en base a las especificaciones de resultados requeridos en las descripciones de casos.

Comprobar la terminación

- Comprobar la terminación normal del proceso de pruebas: Determinar la necesidad de pruebas adicionales en función de los requerimientos o inquietudes planteadas por el historial de fallos de

integridad. En caso de que el proceso haya terminado anormalmente se debe especificar la causa de la contingencia o irregularidad.

- Complementar la prueba: Determinar la necesidad de nuevos casos de pruebas y de ser así añadirlos al conjunto del módulo en prueba.

Evaluar el esfuerzo de la prueba y la unidad

- Registro del estado de las pruebas unitarias: Registrar la cantidad de éxito y fallos de las pruebas ejecutadas.
- Registro del estado de los módulos: Registrar las diferencias reveladas por las pruebas unitarias entre la unidad probada y los requisitos.
- Aseguramiento de la preservación de los productos para pruebas: Asegurar que los productos de pruebas se recojan, organicen y almacenen para la referencia y reutilización.

1.7 Tendencias y tecnologías

Las diferentes etapas que componen el proceso de desarrollo de software basan su funcionamiento sobre diferentes herramientas y tecnologías, alcanzando de esta manera altos índices de productividad, eficiencia y la calidad requerida en el proceso.

1.7.1 Lenguajes de programación

Los lenguajes de programación son lenguajes formales que mediante un conjunto de reglas sintácticas y semánticas permiten crear instrucciones que serán interpretadas por un equipo de cómputo (Terrence y Zelkowitz, 2000). Algunos ejemplos de lenguajes de programación son: PHP, Prolog, Python, Pascal, C++, JAVA, JavaScript.

PHP 5.3

Actualmente el proyecto PRIMICIA se encuentra desarrollando los productos informáticos en el lenguaje de programación PHP en su versión 5.3. Se hace necesario desarrollar las pruebas a aplicar a los módulos seguridad y gestión de medias pertenecientes al sistema XILEMA PRIMICIA 2.0 en el mismo lenguaje utilizado para el desarrollo de este producto para lograr total compatibilidad.

1.7.2 Framework para pruebas unitarias

Las pruebas unitarias permiten detectar errores a gran escala, que con el paso del tiempo requieren mayor inversión para su corrección. Los frameworks juegan un papel fundamental en la ejecución de este tipo de pruebas, en este ámbito se encuentran muchas herramientas de este tipo que permiten realizar seguimiento de pruebas unitarias, acotando el dominio de los frameworks al lenguaje PHP 5.3 y compatibles con el framework de desarrollo Symfony 2.3[2], que se utilizan para desarrollar el sistema XILEMA PRIMICIA 2.0.

En la se caracterizan los principales frameworks de desarrollo que son utilizados para la ejecución de las pruebas unitarias.

Tabla 1: Principales frameworks y sus aspectos principales.

Framework	Descripción	Usabilidad	Características
PHPUnit 4.5.0 (Bergmann, 2015)	PHPUnit 4.5.0 es un framework orientado a la programación de pruebas para PHP. Es un ejemplo de la arquitectura xUnit para marcos de pruebas unitarias.	▪ Soporta objetos Mock[3]. ▪ Soporta Stubs[4]. ▪ Se integra perfectamente a cada entorno de desarrollo integrado siguiente: Eclipse, Netbeans, Zend Stuide, PhpStorm. ▪ Funciona bien con cada servidor de integración continua. ▪ Presenta excelente documentación. ▪ Muy activo respecto a	▪ Soporta el registro de ejecución de la prueba en XML, TAP, o AT & T marcado GraphViz y usando PEAR. ▪ Utiliza aserciones para verificar que el comportamiento de una unidad de código es el esperado. ▪ Compatible con los mapeos objeto-relacional: Doctrine y Propel. ▪ Permite desarrollar pruebas funcionales. ▪ Se integra con Selenium[5] para

[2] Framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web (Potencier y Zaninotto, 2012).

[3] Tipo de objeto doble sobre los se establece expectativas de uso y del que no nos preocupa controlar lo que devuelve su llamada (Bergmann, 2015a).

[4] Tipo de objeto doble que reemplaza una funcionalidad concreta del objeto colaborador para garantizar su funcionamiento (Bergmann, 2015a).

[5] Herramienta que permite escribir pruebas automatizadas de interfaz de usuario para aplicaciones web en cualquier lenguaje de programación en contra de cualquier página web HTTP usando cualquiera navegador convencional (Bergmann, 2015a).

		estado y soporte.	realizar pruebas de aceptación.
SimpleTest 1.1.0 **(Baker, 2015)**	Es un marco de pruebas unitarias PHP y pruebas web. Contiene una interfaz estandarizada a la familia de marcos de pruebas xUnit.	▪ Las pruebas en sí son sólo métodos normales que comienzan su nombre con las letras "test". ▪ Las pruebas se agrupan en casos de prueba, que son sólo clases PHP que extienden UnitTestCase o WebTestCase. ▪ Se puede tener tantos casos de prueba como desee en una secuencia de comandos de prueba. ▪ Presenta documentación básica. ▪ Poco activo respecto ha estado y soporte.	▪ La idea es que las tareas comunes de PHP pero complicados, tales como la tala en un sitio, se pueden probar fácilmente. ▪ Utiliza aserciones para verificar que el comportamiento de una unidad de código es el esperado. ▪ Compatible con los mapeos objeto-relacional: Propel y Doctrine.

Lime (Potencier y Zaninotto, 2012)	Symfony 2 incluye su propio frameowrk llamado Lime. Se basa en la librería Test::More de Perl. Es compatible con TAP, lo que significa que los resultados de las pruebas se muestran con el formato definido en el "Test Anything Protocol", creado para facilitar la lectura de los resultados de las pruebas.	▪ Las pruebas se ejecutan en un entorno de entorno limitado para minimizar las ejecuciones de las pruebas a partir de influenciar uno al otro. ▪ Las pruebas de Lime son fáciles de leer y sus resultados también lo son. ▪ Permite una fácil integración con otras herramientas. ▪ Presenta documentación básica. ▪ Poco activo respecto ha estado y soporte.	▪ Consta únicamente de un archivo, llamado lime.php, y no tiene ninguna dependencia. ▪ El núcleo de Lime se valida mediante pruebas unitarias. ▪ Utiliza aserciones para verificar que el comportamiento de una unidad de código es el esperado. ▪ Compatible con los mapeos objeto-relacional: Doctrine.
Behat 2.4 (Lépine, 2012)	El marco de trabajo para pruebas unitarias Behat permite probar aplicaciones PHP usando oraciones legibles para escribir características y escenarios sobre cómo las aplicaciones deben comportarse con el fin de probar su funcionalidad.	▪ Puede contener una lista de escenarios, los cuales constituyen los posibles estados que toma la unidad a probar (descripción de la situación, un contexto, un acontecimiento, y un resultado). ▪ Presenta poca documentación. ▪ Poco activo respecto ha estado y soporte.	▪ Utiliza una extensión feature. ▪ Utiliza también ¨step¨ los cuales son funciones programadas que definen lo que debe suceder dado una de sus declaraciones se llaman dentro de un escenario.

EnhancePHP (Fenton y Jones, 2011)	Es un marco de pruebas ligero con licencia Open Source desarrollado en PHP. Las sintaxis de las pruebas unitarias son simples y natural. Constituyes una gran biblioteca en un solo archivo PHP, que ofrece una gran cantidad de funcionalidades de pruebas.	▪ Soporta objetos Mock. ▪ Soporta Stubs. ▪ Rápido para empezar ▪ Informes en diferentes formatos (CLI, HTML, TAP, XML) ▪ Los resultados de cobertura de código son sólo a nivel de método. ▪ No se han producido mejoras desde 2013. ▪ Presenta poca documentación. ▪ Poco activo respecto ha estado y soporte.	▪ Consta únicamente de un archivo PHP. ▪ Define el objetivo para la clase a probar, a continuación se establecen escenarios para realizar varias llamadas a un mismo método y luego se realiza la verificación de la expectativa de los escenarios.
Atoum 1.0.0 (Croes, 2014).	Es un marco de trabajo para pruebas unitarias específicas del lenguaje PHP. Fue diseñado para ser aplicado rápidamente, simplificar el desarrollo de las pruebas además de garantizar su fiabilidad y legibilidad.	▪ Requiere PHP 5.3.3. ▪ Soporta objetos Mock. ▪ Soporta Stubs. ▪ Fácil de instalar. ▪ Compatible con Symfony 2 y Zend Framework 2. ▪ Genera informes de cobertura de código, con el fin de hacer posible para supervisar las pruebas unitarias. ▪ Presenta poca documentación. ▪ Activo respecto ha estado y soporte.	▪ Utiliza aproximadamente 200 aserciones para verificar que el comportamiento de una unidad de código es el esperado. ▪ Consta únicamente de un solo archivo Phar. ▪ Permite escribir las pruebas unitarias de una manera cerca de lenguaje natural.

Tras haber sintetizado el contenido de la tabla comparativa anterior se selecciona como framework para el desarrollo de las pruebas unitarias el PHPUnit 4.5.0. Esta herramienta prácticamente se ha convertido en

un estándar internacional para el desarrollo de pruebas para PHP. Las pruebas unitarias de Symfony 2.3 combinan la potencia de PHPUnit 4.5.0 con las utilidades y facilidades proporcionadas por Symfony 2.3 (Eguiluz, 2012). Un aspecto importante es que PHPUnit es el único framework que ha demostrado prácticamente contar con una gran cobertura de código, estabilidad y buena documentación (Bahit, 2013). Además de permitir realizar otros tipos de pruebas automatizadas complementando de esta manera el proceso.

1.7.3 Entorno de desarrollo integrado

Dentro del conjunto de herramientas que sustentan la producción de software, los entornos de desarrollo integrado (IDE) son de gran importancia, ya que suman funcionalidades y artefactos claves para acelerar el proceso de desarrollo del ciclo de vida del software. Un IDE es un programa informático compuesto por un conjunto de herramientas para programar en un lenguaje de programación o varios. Las herramientas que normalmente componen un entorno de desarrollo integrado son: editor de texto, compilador, intérprete, herramientas para la automatización, depurador, sistema de ayuda para la construcción de interfaces gráficas de usuario y, opcionalmente, un sistema de control de versiones (Guest, Greenfield y Cook, 2006).

NetBeans IDE

El entorno de desarrollo integrado NetBeans 8.0 fue seleccionado para el desarrollo de la propuesta informática por disímiles características y facilidades en su uso. Un aspecto importante es la experiencia y conocimiento del equipo de desarrollo con la herramienta por lo que se gana en competencias del personal. Además de permitir total integración con Symfony 2.3 y PHPUnit 4.5 aspecto de suma importancia y garantía.

NetBeans IDE permite de forma rápida y fácil desarrollar aplicaciones Java de escritorio, móviles y aplicaciones Web, así como aplicaciones HTML5 con HTML, JavaScript y CSS. El IDE también proporciona un gran conjunto de herramientas para desarrolladores de PHP y C / C ++. Es gratuito y de código abierto. (NetBeans Team, 2015).

1.8 Conclusiones parciales

Luego de realizar la revisión teórica de los conceptos y definiciones relacionados con la calidad de software, aseguramiento y factores de medición de la misma, las pruebas de software (haciendo énfasis en las pruebas unitarias), las tecnologías y herramientas que rigen el adecuado desarrollo de software se concluye lo siguiente:

- Las pruebas unitarias permiten evaluar el software de manera aislada, es decir por módulos, chequeando partes del código y permitiendo mejor exactitud en la detección del código erróneo y su temprana corrección, constituyendo ventajas significativas para la calidad del producto.
- El proceso de pruebas unitarias requiere para su exitoso funcionamiento la total integración y compatibilidad de las herramientas del entorno de producción del proyecto con las seleccionadas para desarrollar las pruebas unitarias.
- Mediante el uso del framework de pruebas unitarias PHPUnit 4.5.0 se garantiza la automatización de estas, se agiliza el proceso y favorece la detección de no conformidades

Capítulo 2: Planificación, diseño y desarrollo

2.1 Introducción

En el presente capítulo se exponen los principales procesos y artefactos generados en las etapas de planificación, diseño y desarrollo del proceso de pruebas unitarias definido en el estándar (IEEE Std. 1008-1987). Lo anterior tiene como objetivo lograr la implementación de un conjunto de pruebas unitarias referentes a los módulos seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0 y de esta manera contribuir a la calidad tanto del producto como del ciclo de vida del desarrollo de software.

2.2 Planificación del enfoque general, recursos y calendario

En esta etapa se debe obtener el esfuerzo y tiempo necesario para las pruebas, productos entregables, recursos requeridos, repositorios y ambientes de prueba entre otros aspectos.

2.2.1 Enfoque general

Para el proceso de automatización de las pruebas es necesario realizar el enfoque general del proceso ya que este constituye la base y punto de partida para las diferentes actividades que componen el marco del proceso. Integrando las técnicas de diseño e implementación de las pruebas con un conjunto de actividades que complementan la estrategia puesta en práctica.

El método de prueba a utilizar en el proceso de automatización de las pruebas aplicado a los módulos seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0 es el de Caja Blanca[6], con el fin de estudiar la especificación de las funciones o métodos, la entrada y la salida, definiendo como objetivo fundamental probar las posibles entradas y salidas de los métodos permitiendo cubrir el mayor porcentaje de código probado por pruebas unitarias (cobertura de código[7]).

[6] Método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba (Pressman, 2002).

[7] Medida utilizada para describir el grado en el que el código fuente de un programa es probado por un conjunto de pruebas en particular (Bergmann 2015a).

Mediante la tecnología del marco de trabajo PHPUnit 4.5.0 se desarrollará una colección de pruebas unitarias. De esta colección se recogerán datos de interés mediante los informes generados por el marco de trabajo.

2.2.2 Especificación de los requisitos de integridad

El sistema PRIMICIA 2.0 utiliza la arquitectura modelo vista controlador (MVC) la cual separa los diferentes tipos de clases, la colección de pruebas abarcará las clases los tipos de clases que se prueben mediante pruebas unitarias. Agregando a esto que se requiere de un grado de cobertura de código alto en cada caso además de la totalidad de los métodos probados.

2.2.3 Especificación de los requisitos de terminación

Es necesario establecer determinados requisitos o hitos que midan el fin del proceso de pruebas, en este caso el cubrimiento total de clases programadas que se prueben con efectividad mediante pruebas unitarias de los módulos seguridad y gestión de medias, además de seguir la planificación de las actividades. La principal condición a cumplirse por la terminación anormal del proceso es el incumplimiento del calendario por lo que no se terminaría a tiempo el producto.

2.2.4 Recursos para las pruebas unitarias

Los recursos para aplicar las pruebas unitarias automatizadas a los módulos seleccionados del sistema XILEMA PRIMICIA 2.0, son características tanto de software como de hardware. Para el desarrollo de las pruebas es necesario disponer de los siguientes recursos mínimos:

Tabla 2: Recursos de Software.

Software	
Recurso	**Especificación**
Sistema Operativo.	GNU/Linux.
Sistema a probar.	XILEMA PRIMICIA 2.0.
Servidor Web.	Apache 2.
Marco de Trabajo.	PHPUnit 4.5.0.

Tabla 3: Recursos de Hardware.

Hardware	
Recurso	**Especificación**
Procesador.	Core 2 Duo 3.3 GHz.
Memoria RAM.	1 Giga Byte (Gb) o más.
Disco Duro.	120 Giga Byte (Gb) o más.

2.2.5 Cronograma general

Es de suma importancia establecer un calendario que guíe y permita controlar las diferentes tareas a realizar lo más realista posible. A continuación se muestra el cronograma general del proceso.

Tabla 4: Cronograma del proceso de pruebas unitarias.

Fase	Actividad	Tareas	Fecha Inicio	Fecha Fin
1	Planificar el enfoque general, recursos y calendario.	Especificar un enfoque general de las pruebas unitarias.	17/11/2014	18/11/2014
		Especificar los requisitos de integridad.	19/11/2014	20/11/2014
		Especificar los requisitos de terminación.	21/11/2014	22/11/2014
		Determinar las necesidades de recursos.	23/11/2014	24/11/2014
	Determinar características para ser probados.	Estudiar los requisitos funcionales.	25/11/2014	28/11/2014
	Refinar el plan general.	Acotar el enfoque.	29/11/2014	30/11/2014
		Especificar las necesidades de recursos especiales.	1/12/2014	2/12/2014
2	Diseñar el conjunto de pruebas.	Diseñar la arquitectura de la prueba.	3/12/2014	7/12/2014
		Obtener las especificaciones de casos de prueba.	8/12/2014	20/12/2014
	Implementar el plan y el diseño refinado.	Obtener y verificar datos de prueba.	8/01/2015	11/01/2015
		Obtener recursos especiales.	12/01/2015	13/01/2015
		Desarrollar los casos de pruebas.	14/01/2015	13/04/2015
3	Ejecutar los procedimientos de prueba.	Ejecutar pruebas.	14/01/2015	16/04/2015
		Determinar los resultados.	17/04/2015	19/04/2015
	Comprobar la terminación.	Comprobar la terminación normal del proceso de pruebas.	20/04/2015	21/04/2015
		Complementar la prueba.	22/04/2015	26/04/2015
	Evaluar el esfuerzo de la	Registro del estado de las pruebas unitarias.	27/04/2014	28/04/2014

	prueba y la unidad.	Registro del estado de los módulos.	29/04/2014	30/04/2014
		Asegurar la preservación de los productos para pruebas.	1/05/2014	2/05/2014
4	Evaluación de los resultados de las pruebas unitarias luego de aplicarlas para los módulos de seguridad y gestión de medias del sistema PRIMICIA 2.0.		3/05/2014	10/05/2014

2.3 Especificación de las características a ser probadas

En este proceso de pruebas es de suma importancia analizar los requerimientos o requisitos funcionales del sistema a probar en este caso XILEMA PRIMICIA 2.0. A continuación se muestra el paquete de análisis del sistema XILEMA PRIMICIA 2.0, en el cual se pueden enmarcar los módulos referentes al proceso de pruebas.

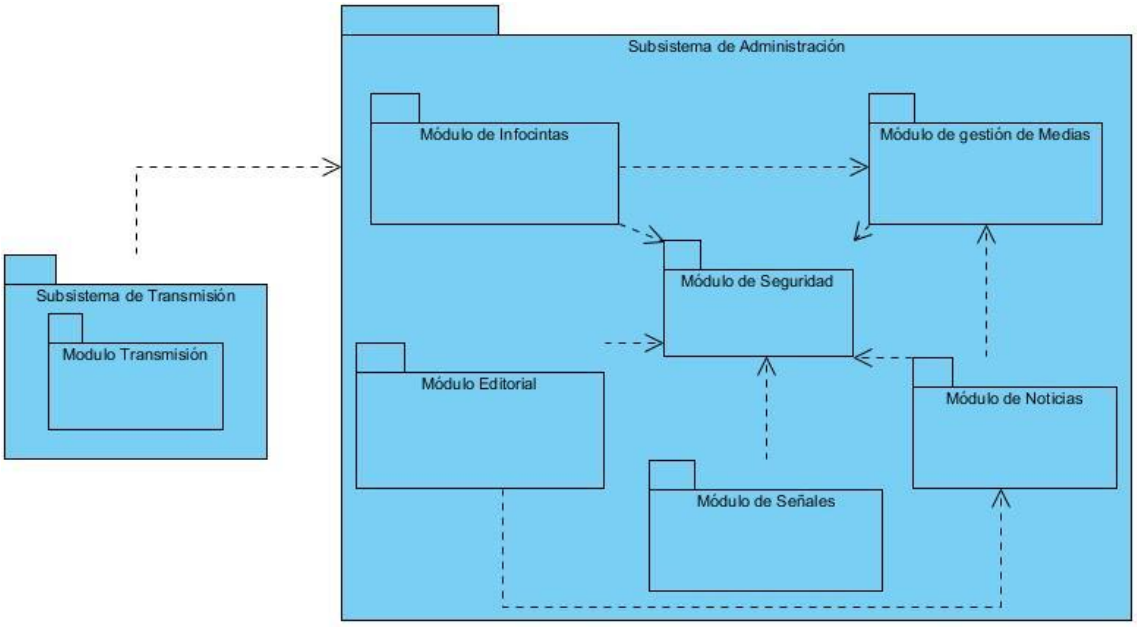


Figura 4: Paquete de análisis del sistema PRIMICIA 2.0 (PRIMICIA, 2013).

2.3.1 Modulo seguridad

El módulo de seguridad es el encargado de realizar toda la gestión de los usuarios, grupos y permisos del sistema. Debe permitir la adición, modificación y eliminación de los usuarios que trabajarán en el sistema. Además realiza la gestión de los grupos de permisos que se podrán asignar a los usuarios para darles acceso a las funcionalidades (PRIMICIA, 2012b).

Tabla 5: Especificación de requisitos de software del módulo seguridad (PRIMICIA, 2013).

Nº	Nombre	Descripción
RF1	Autenticar usuario	Para su acceso al sistema el usuario debe introducir su usuario y contraseña y seleccionar si desea recordar los datos introducidos. Esta acción le permite acceder a las funcionalidades correspondientes al rol que posee el usuario autenticado.
RF2	Cerrar sesión	El sistema debe permitir cerrar la sesión de un usuario autenticado.
RF3	Listar usuario	El sistema debe permitir que se muestre un listado con los usuarios del sistema.
RF4	Insertar usuario	El sistema debe permitir que se inserten nuevos usuarios.
RF5	Editar usuario	El sistema debe permitir que se realicen cambios en los datos y permisos de los usuarios.
RF6	Eliminar usuario	El sistema debe permitir que se eliminen los usuarios que sean seleccionados.
RF7	Filtrar usuarios	El sistema debe permitir que se filtren los usuarios teniendo en cuenta los datos introducidos.
RF8	Listar grupos	El sistema debe permitir que se muestre un listado con los grupos que se encuentran en el sistema.
RF9	Insertar grupo	El sistema debe permitir que se inserten nuevos grupos.
RF10	Editar grupo	El sistema debe permitir que se realicen cambios en los datos y permisos de los grupos.
RF11	Eliminar grupos	El sistema debe permitir que se eliminen grupos.
RF12	Filtrar Grupos	El sistema debe permitir que se filtren los grupos teniendo en cuenta los datos introducidos.
RF13	Listar Permisos	El sistema debe permitir que se muestre un listado con los permisos que se pueden asignar en el sistema.
RF14	Filtrar permisos	El sistema debe permitir que se filtren los permisos teniendo en cuenta los datos introducidos.

2.3.2 Modulo gestión de medias

El módulo de gestión de medias es el encargado de realizar toda la gestión de los archivos multimedia (audio, video e imagen) que pueden ser utilizados en el sistema para la creación de las informaciones. Debe permitir la adición, modificación, visualización y/o reproducción y eliminación de estos archivos multimedia (PRIMICIA 2012a).

Tabla 6: Especificación de requisitos de software del módulo gestión de medias (PRIMICIA, 2013).

Nº	Nombre	Descripción
RF15	Visualizar imagen	Se debe mostrar la imagen seleccionada, además todos los datos relacionados con la imagen que se visualiza.
RF16	Listar imágenes	Debe permitir listar los archivos de imágenes almacenados.
RF17	Filtrar imágenes	Debe permitir filtrar las imágenes existentes en el sistema, teniendo en cuenta los criterios de búsqueda introducidos.
RF18	Adicionar imagen	Debe permitir adicionar nuevas imágenes al sistema.
RF19	Modificar imagen	Debe permitir modificar los datos de las imágenes del sistema.
RF20	Eliminar imágenes	Debe permitir eliminar la (s) imagen (es) deseada (s) del sistema.
RF21	Reproducir audio	Debe reproducir el audio seleccionado, además se deben mostrar los datos del mismo.
RF22	Pausar reproducción del audio	Debe permitir que se ponga en pausa el audio que se está reproduciendo.
RF23	Detener reproducción del audio	Debe permitir que se pare el audio que se está reproduciendo.
RF24	Modificar volumen de audio	El sistema debe permitir que se suba o se baje el volumen del audio que se esté reproduciendo.
RF25	Listar audios	El sistema debe permitir listar los archivos de audio almacenados.
RF26	Filtrar audios	Debe permitir filtrar los archivos de audio existentes en el sistema, teniendo en cuenta los criterios de búsqueda introducidos.
RF27	Adicionar audio	Debe permitir adicionar nuevos archivos de audio al sistema.
RF28	Modificar audio	Debe permitir modificar los datos de los audios que se encuentran en el sistema.
RF29	Eliminar audios	Debe permitir eliminar el (los) audio (s) deseado (s) del sistema.
RF30	Reproducir video	Se debe reproducir el video seleccionado, además se deben mostrar todos los datos relacionados con este.
RF31	Pausar reproducción del video	El sistema debe permitir que se ponga en pausa el video que se está reproduciendo.
RF32	Detener reproducción del video	El sistema debe permitir que se pare el video que se está reproduciendo.
RF33	Modificar volumen de video	El sistema debe permitir que se suba o se baje el volumen del video que se esté reproduciendo.

RF34	Listar videos	El sistema debe permitir listar los videos almacenados.
RF35	Filtrar videos	Debe permitir filtrar los archivos de video existentes en el sistema, teniendo en cuenta los criterios de búsqueda introducidos.
RF36	Adicionar video.	Debe permitir adicionar nuevos archivos de video al sistema.
RF37	Modificar video.	Debe permitir modificar los datos de los videos que se encuentran en el sistema.
RF38	Eliminar videos.	Debe permitir eliminar el (los) video (s) deseado (s) del sistema.

2.3.3 Requisitos adicionales

Para el sistema XILEMA PRIMICIA 2.0 es sumamente importante controlar algunos requisitos no funcionales. A continuación se presentan los requisitos adicionales que se controlan de manera efectiva mediante las pruebas unitarias.

- Controlar picos de uso de memoria.
- Leer picos de memoria por petición.
- Tiempo de respuesta.

2.4 Diseño de la colección casos de pruebas unitarias

Esta etapa es de suma importancia por constituir la base de la implementación de la colección de pruebas unitarias, donde se define la arquitectura, notación y casos de pruebas unitarias.

2.4.1 Diseño de la arquitectura de pruebas unitarias

Cada módulo del sistema XILEMA PRIMICIA 2.0 está compuesto por una estructura de directorios, dentro de la misma se encuentra el directorio "Test". Este directorio debe tener reproducida la misma estructura de directorios del módulo que se quiere probar y aquí es donde se almacenan las pruebas unitarias. Los archivos auxiliares que necesitan las pruebas unitarias se guardan en el directorio "Fixture" dentro del mismo directorio donde se encuentra la prueba unitaria.

Para construir mejores y más confiables aplicaciones, se debe probar el código usando ambas pruebas, unitarias y funcionales. Por lo tanto las pruebas funcionales automatizadas se encargan de probar las clases controladoras (Potencier y Weaver, 2013). En la estructura de directorio creada se dejó listo el

directorio denominado "Controller" para almacenar las pruebas funcionales realizadas por el personal del proyecto PRIMICIA en su etapa correspondiente. Este directorio se muestra en la Figura 5.

Cada prueba unitaria se define en una clase donde su nombre coincide con el nombre de la clase a probar, agregándosele la terminación Test. Cada prueba unitaria comienza su nombre con la palabra test y luego un nombre descriptivo del método que se está probando en notación *"camel case".* Un ejemplo se muestra en la Figura 6.

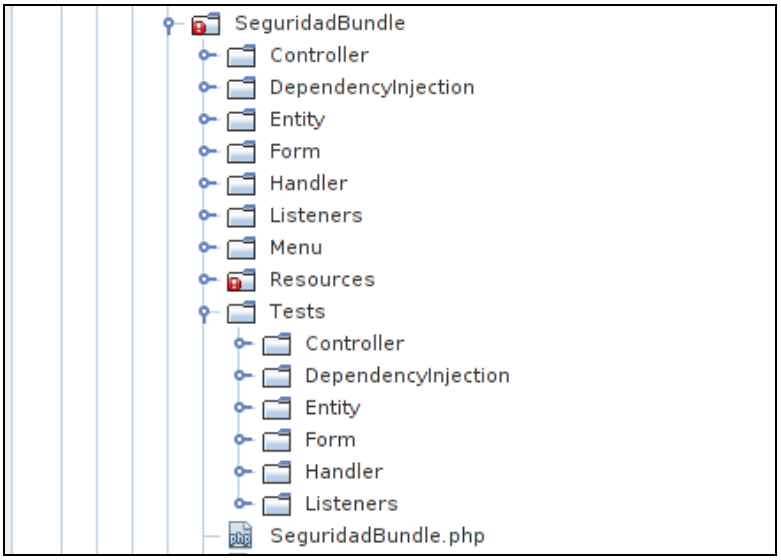


Figura 5: Estructura de directorios. Modulo seguridad.

```php
namespace Primicia\SeguridadBundle\Tests\Entity;

use Primicia\SeguridadBundle\Entity\Usuario;

/**
 * Description of UsuarioTest
 *
 * @author dairon
 */
class UsuarioTest extends \PHPUnit_Framework_TestCase {

    public function testIsActiveTrue() {

    }
```

Figura 6: Notación de clases y pruebas.

2.4.2 Especificación de casos de pruebas unitarias

En la especificación de casos de pruebas unitarias se debe tener en cuenta que cada caso de prueba se realiza por clases del código bajo prueba. El modelo fue realizado por un grupo de trabajo de conjunto con los autores.

A continuación se muestra una ejemplificación de los casos de pruebas que contienen alta prioridad para el cliente, pertenecientes a los módulos seguridad y gestión de medias.

Tabla 7: Caso de pruebas unitarias. Modulo Seguridad. Clase UsuarioRepository.

Casos de Prueba de Unidad					
Código del Caso de Prueba: CPU-UsuarioRepositoryTest					
Iteración: 1.0.					
Modulo: Seguridad.					
Clase: UsuarioRepository.					
Descripción de la prueba: Se realizan pruebas de unidad referentes a los métodos de la clase UsuarioRepository perteneciente al módulo seguridad de la plataforma XILEMA PRIMICIA 2.0.					
Nombre del encargado:					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado esperado**	**Resultado de la prueba**	**Observaciones**
findunBoundGroups	testfindunBound	String,	array	Fallo	-

	GroupsFalse()	GroupManager, UserManager			
	testfindunBound GroupsTrue()	String, GroupManager, UserManager	-	Fallo	-
	testfindunBoundGroups ArraysUserAndGroups OneObjectDistinct()	String, GroupManager, UserManager	array	Éxito	-
bindGroup	testbindGroupNotAdd()	GroupManager, UserManager, String, String	False	Fallo	-
	testbindGroupOkAdd()	GroupManager, UserManager, String, String	False	Éxito	-
DeleteMuch	testDeleteMuchIdEmpty()	String	False	Éxito	-
	testDeleteMuchIds()	String	True	Éxito	-
	testDeleteMuchException()	String	Exception	Éxito	-

Tabla 8: Caso de pruebas unitarias. Modulo Gestión de medias. Clase AudioRepository.

Casos de Prueba de Unidad					
Código del Caso de Prueba: CPU-AudioRepositoryTest					
Iteración: 1.0.					
Modulo: Gestión de medias.					
Clase: AudioRepository.					
Descripción de la prueba: Se realizan pruebas de unidad referentes a los métodos de la clase AudioRepository perteneciente al módulo gestión de medias de la plataforma XILEMA PRIMICIA 2.0.					
Nombre del encargado:					
Funcionalidad	**Método utilizado**	**Recibe**	**Resultado esperado del método**	**Resultado de la prueba**	**Observaciones**
DeleteMuch	testDeleteMuchIdEmpty()	Integer	Boolean	Éxito	-

	testDeleteMuchIds()	Integer	Boolean	Éxito	-
	testDeleteMuchException()	Integer	Boolean	Fallo	-
isDeletable	testIsDeletableNoticeNull()	Integer	Boolean	Éxito	-
	testIsDeletableNoticeObject()	Integer	Boolean	Éxito	-

2.5 Implementación

La etapa de implementación es de suma importancia en el proceso ya que es fundamental en la obtención de resultados satisfactorios.

2.5.1 Obtención y verificación de datos de pruebas

El proyecto que desarrolla el sistema XILEMA PRIMICIA 2.0 no cuenta con datos, ya que a esta versión actual aún no se le ha realizado ningún tipo de pruebas debido a dos razones fundamentales, el calendario y la fase de la metodología en que se encuentra. Por lo tanto solo se pudo obtener el código fuente del sistema actualmente en desarrollo y la documentación referente a los módulos que presentan su código bajo pruebas.

2.5.2 Obtención de recursos especiales

Las pruebas unitarias se realizan en el ambiente de producción por lo que se hace indispensable que el sistema cuente con los mismos recursos que en su desarrollo. Producto de esto, se instalan las tecnologías y herramientas del entorno de desarrollo del sistema.

2.5.3 Desarrollo de los casos de prueba

Se implementaron los casos de prueba diseñados en las fases anteriores. Cuidando siempre la integridad de los datos de cada función y del código obtenido referente al sistema XILEMA PRIMICIA 2.0.

A continuación se muestran una selección de diferentes clases que contienen las pruebas unitarias almacenadas, resaltando las principales técnicas utilizadas para la confección de las mismas.

La Figura 7 muestra una ejemplificación de la utilización de los diferentes métodos de aserciones accesibles para comprobación de los resultados esperados con los reales, pertenecientes al framework de pruebas PHPUnit 4.5.0.

```
    /**
     * Pruebas unitarias automatizadas
     */
    public function testGetUbicacion() {
        $this->assertNull($this->audio->getUbicacion());
    }

    /**
     * @depends testGetUbicacion
     */
    public function testSetUbicacion() {
        $dir = 'Primicia\MediasBundle\Tests\Entity\Fixture';
        $this->audio->setUbicacion($dir);
        $this->assertNotNull($this->audio->getUbicacion());
        $this->assertEquals($dir, $this->audio->getUbicacion());
    }
```

Figura 7: Utilización de los métodos de aserciones de PHPUnit 4.5.0.

La Figura 8 muestra una ejemplificación de la utilización del método setUp() definido por el framework de pruebas PHPUnit 4.5.0 para ejecutarse antes de cada prueba y establecer la condiciones iniciales requeridas, además de contribuir significativamente a la reutilización de código.

```
    protected function setUp(){
        $emMock = $this->getMockBuilder('Doctrine\ORM\EntityManager')
                ->setMethods(array())
                ->disableOriginalConstructor()
                ->getMock();

        $classMock = $this->getMockBuilder('Doctrine\ORM\Mapping\ClassMetadata')
                ->setMethods(array())
                ->disableOriginalConstructor()
                ->getMock();
        $this->userRepo = new UsuarioRepository($emMock, $classMock);
    }
```

Figura 8: Utilización del método setUp.

La Figura 9 muestra una ejemplificación de la notación utilizada para generar dependencias explícitas de pruebas que necesitan de la ejecución exitosas de otras.

```
public function testGetDisponible() {...3 lines }

/**
 * @depends testGetDisponible
 */
public function testSetDisponibleTrue() {...4 lines }

/**
 * @depends testGetDisponible
 */
public function testSetDisponibleFalse() {...4 lines }

/**
 * @depends testGetDisponible
 * @depends testSetDisponibleTrue
 */
public function testSetDisponibleNull() {...5 lines }

public function testGetGenerica() {...3 lines }

/**
 * @depends testGetGenerica
 */
public function testSetGenericaTrue() {...4 lines }

/**
 * @depends testGetGenerica
 */
public function testSetGenericaFalse() {...4 lines }
```

Figura 9: Utilización de las dependencias de pruebas.

La Figura 10 muestra una ejemplificación de la utilización de las pruebas-dobles (Objetos Mock y Stubs) utilizados para simular las diferentes dependencias de los métodos bajo pruebas y lograr la independencia de métodos en prueba.

```php
public function testBindRolEmptyRol() {
    $name = $this->generateLetersChain(rand(1, 250));

    $groupMock = $this->getMockBuilder('Primicia\SeguridadBundle\Entity\Grupo')
            ->setMethods(array())
            ->disableOriginalConstructor()
            ->getMock();
    $groupMock->expects($this->never())
            ->method('hasRole')
            ->with();

    $groupManagerMock = $this->getMockBuilder('FOS\UserBundle\Doctrine\GroupManager')
            ->setMethods(array('findGroupByName'))
            ->disableOriginalConstructor()
            ->getMock();
    $groupManagerMock->expects($this->once())
            ->method('findGroupByName')
            ->with($this->identicalTo($name))
            ->will($this->returnValue($groupMock));

    $this->assertFalse($this->groupRepo->bindRol($groupManagerMock, $name, ''));
}
```

Figura 10: Utilización de pruebas-dobles.

La Figura 11 muestra una ejemplificación de la utilización proveedores de datos que brinda el framework PHPUnit 4.5.0, permitiendo mejor funcionalidad e incidiendo sobre la tarea tediosa de ejecutar las pruebas con muchos datos de uno en uno.

```php
/**
 * @depends testGetMediaType
 * @dataProvider additionProvider
 */
public function testSetMediaType($expectedValue, $mediaType) {
    $mediaHandler = new MediaHandler(NULL, NULL, NULL, NULL, NULL);
    $mediaHandler->setMediaType($mediaType);
    $this->assertEquals($expectedValue, $mediaHandler->getMediaType());
}

public function additionProvider() {
    return array(
        array('imagen', 'imagen'),
        array('video', 'video'),
        array('audio', 'audio'),
    );
}
```

Figura 11: Utilización de proveedores de datos.

2.6 Ejecución

Constituye la base de todo un estudio en torno a la mejoría de la calidad del sistema que presenta el código bajo pruebas y diferentes reportes que potencian la documentación tanto del ciclo de vida del sistema como del proceso de pruebas unitarias. A continuación se muestra el flujo de control de esta actividad que guía esta fase del proceso.

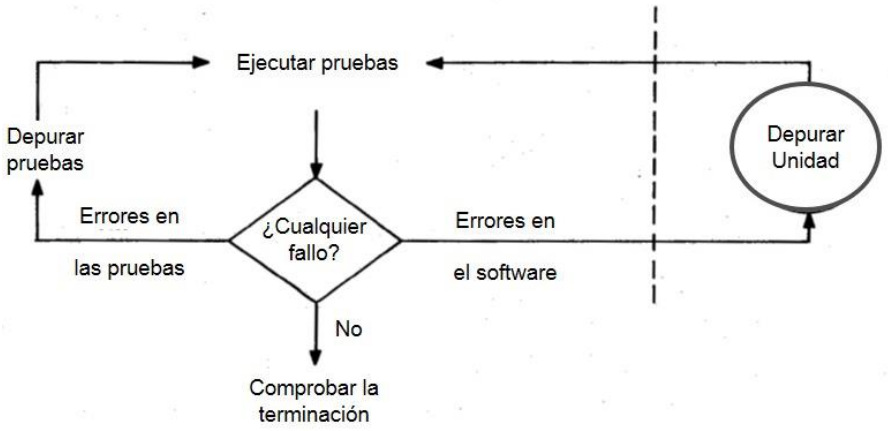


Figura 12: Control del flujo de actividad de la fase Ejecución (IEEE 1993).

2.6.1 Ejecución de la colección de pruebas unitarias

Se ejecutó el conjunto de casos de prueba implementados y se registraron los incidentes arrojados en el plan de pruebas confeccionado al inicio del proceso y actualizado durante todas las actividades realizadas de dicho proceso.

Las figuras 13, 14 y 15 muestran la ejecución del conjunto de pruebas implementadas a los módulos seguridad, gestión de medias y la colección de pruebas, respectivamente, mediante las opciones de línea de comandos[8]. Identificándose de esta manera el estado de cada prueba, el tiempo de ejecución, la memoria que fue utilizada y una breve especificación en caso de constituir error; además de datos de generales de la ejecución como son la cantidad de pruebas, de aserciones y de errores.

[8] phpunit -c app <ubicación del scrip de pruebas>.

Figura 13: Ejecución de las pruebas unitarias. Módulo seguridad.

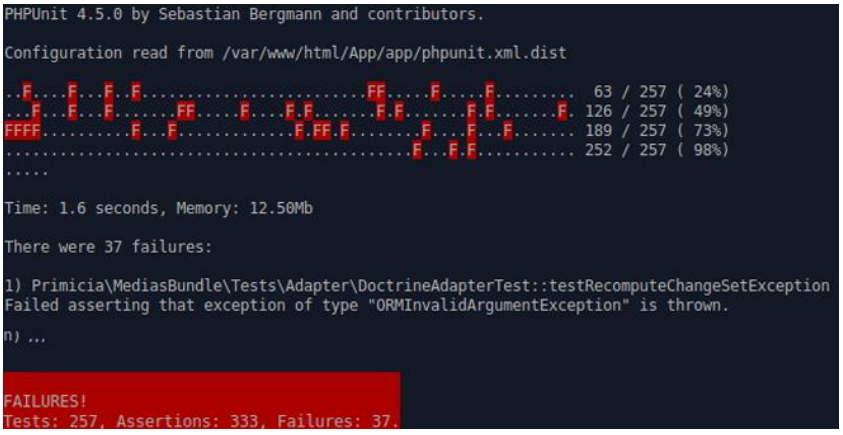


Figura 14: Ejecución de las pruebas unitarias. Módulo gestión de medias.

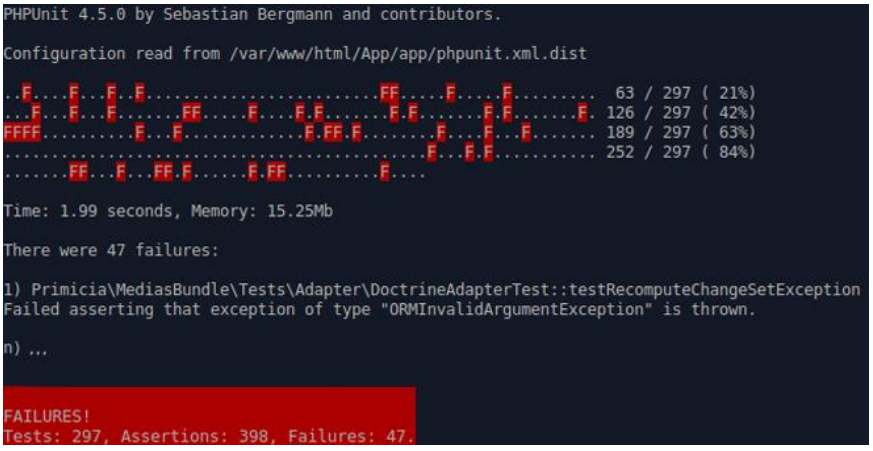


Figura 15: Ejecución de las pruebas unitarias. Colección completa.

2.6.2 Determinar resultados

Se determina el grado de afectación de los errores arrojados por la ejecución de la colección de pruebas desarrolladas. Esta clasificación se distribuye en grados de baja, media o crítica, teniendo en cuenta la prioridad para el cliente de la funcionalidad asociada y su nivel de complejidad.

Según el grado asociado se asigna la acción a tomar en el área de producción dando prioridad a los elementos críticos.

A continuación se muestran los reportes de errores.

Tabla 9: Reporte de errores. Modulo seguridad.

Reporte de Errores			
Errores encontrados: 10.			
Funcionalidad: Filtrar permisos.			
Método	**Error**	**Clasificación**	**Observaciones**
BindRol()	Presenta problemas al trabajar con lista de errores vacías.	Bajo	Refactorizar
BindRol()	Presenta problemas al devolver los valores esperados cuando no se ejecuta la acción.	Bajo	Refactorizar
Funcionalidad: Filtrar usuarios.			
IsUsed()	Presenta problemas al devolver los valores esperados cuando un usuario se encuentra en el grupo especificado.	Bajo	Refactorizar
Funcionalidad: Filtrar Grupos.			
findunBoundGroups()	Presenta problemas al retornar que el grupo no tiene asociado el usuario especificado.	Bajo	Refactorizar
findunBoundGroups()	Presenta problemas al retornar que el grupo tiene asociado el usuario especificado.	Bajo	Refactorizar
Funcionalidad: Editar usuario.			
bindGroup()	Presenta problemas al retornar los valores booleanos correspondientes, retorna el mismo valor para todo camino tomado en la estructura condicional.	Medio	Refactorizar inmediatamente

setActive()	Presenta problemas al editar los datos de la entidad.	Medio	Refactorizar inmediatamente
setActive()	Presenta problemas al editar los datos de la entidad.	Medio	Refactorizar inmediatamente
setActive()	Presenta problemas al editar los datos de la entidad.	Medio	Refactorizar inmediatamente
Funcionalidad: Listar Permisos.			
getRoleList()	Presenta problemas al devolver los roles listados en una estructura de datos array.	Bajo	Refactorizar

2.7 Comprobación

La comprobación es vital tanto para la calidad del sistema XILEMA PRIMICIA 2.0 del cual su código está bajo pruebas como para la calidad del proceso de pruebas unitarias, dado que el mismo constituye un suplemento al conjunto de pruebas. A continuación se muestra el flujo de control de esta actividad que guía esta fase del proceso.

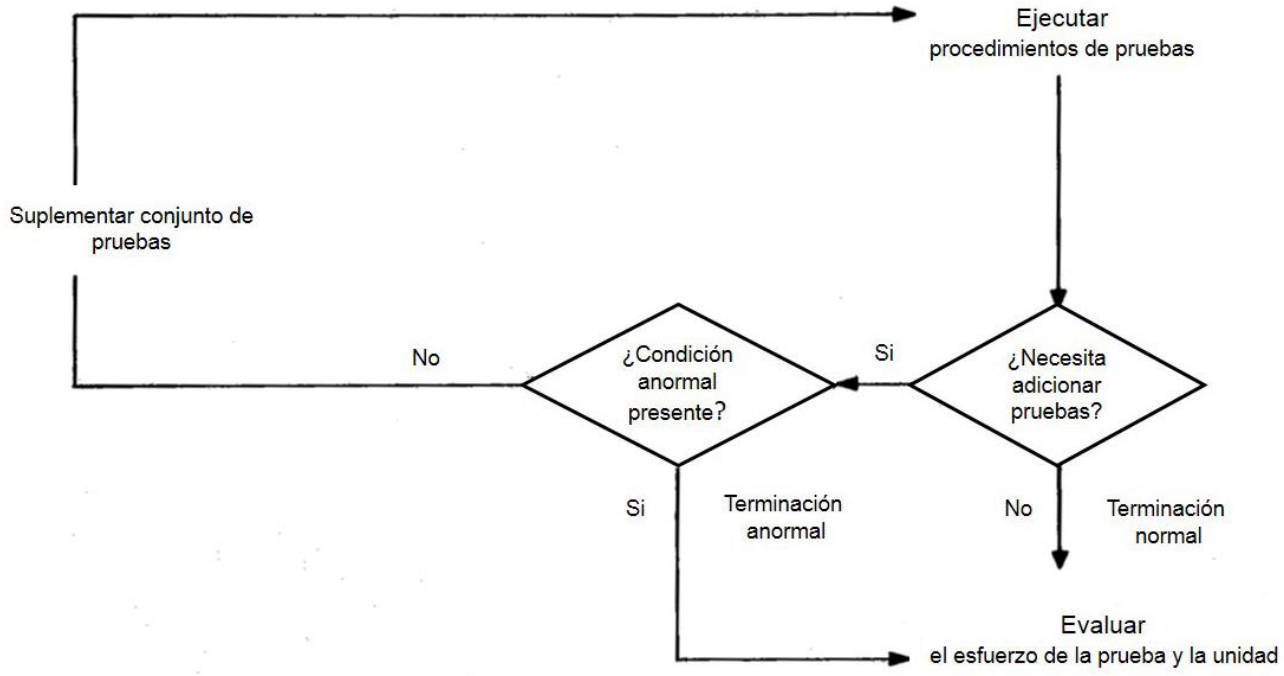


Figura 16: Control del flujo de actividad de la fase Comprobación (IEEE 1993).

2.7.1 Comprobación de la terminación normal del proceso de pruebas

Según el calendario del proceso trazado en las primeras actividades del proceso de pruebas unitarias, este concluye normalmente ajustándose al cronograma. La colección de pruebas unitarias automatizada desarrollada contiene altos índices de cobertura de código lo que permite asegurar que se probaron los diferentes flujos de control de los métodos de las clases bajo prueba.

2.7.2 Complementar el conjunto de pruebas unitarias

Se comprobaron minuciosamente todos los casos de prueba, se añadieron nuevos casos de prueba necesarios, se señala que las condiciones anormales de terminación (incumplimiento del calendario o la incapacidad de cubrir el código bajo pruebas) no se manifestaron. Además, se identificaron nuevos casos de prueba que complementan la colección de pruebas existente, luego se procedió a su debida documentación e implementación. Esta actividad constituye un enlace iterativo e incremental con las actividades anteriores.

2.8 Conclusiones parciales

Luego de materializar satisfactoriamente las diferentes actividades que componen las fases del proceso de pruebas unitarias, exceptuando la evaluación, se concluye lo siguiente:

- La etapa de planificación de pruebas unitarias es de suma importancia para la terminación normal del proceso.
- El diseño y la implantación forman la fase de pruebas adquiridas, la cual constituye el eje central del proceso. Esta fase incide de manera directa sobre la integración del proceso a la producción de software, el éxito del mismo y de manera colateral en la calidad del producto.
- Las etapas de ejecución y comprobación deben ser tenidas siempre en cuenta ya que tienen un alto valor de apoyo y suplemento a la fase de pruebas adquiridas, además de determinar aspectos fundamentales de interés.
- La verificación del correcto funcionamiento luego de la corrección del software se realiza automáticamente con la ejecución de los casos de prueba implementados anteriormente, cada vez que se necesite o se actualice el software del sistema.

Capítulo 3: Evaluación

3.1 Introducción

En el presente capítulo se exponen las actividades realizadas en la etapa de evaluación, última del proceso de pruebas unitarias. Además, se realiza el estudio correspondiente luego de terminado el proceso para determinar las posibles conclusiones de su integración y puesta en práctica como parte del proceso de desarrollo de software. Dicho estudio genera datos estadísticos de los módulos bajo pruebas e información de interés con respecto a análisis establecidos, tomándose como base información de proyectos terminados y tendencias del año 2014 del Centro GEYSED.

3.2 Evaluación

La actividad de evaluación del proceso definido en el estándar IEEE Std. 1008-1987 es determinante al momento de inclinarse por su integración al proceso de desarrollo dado que se registran los resultados. También constituye un método de concientización del personal desarrollador de aplicaciones informáticas, solventando una de las principales dificultades de la aceptación y la puesta en práctica del proceso de automatización de pruebas unitarias.

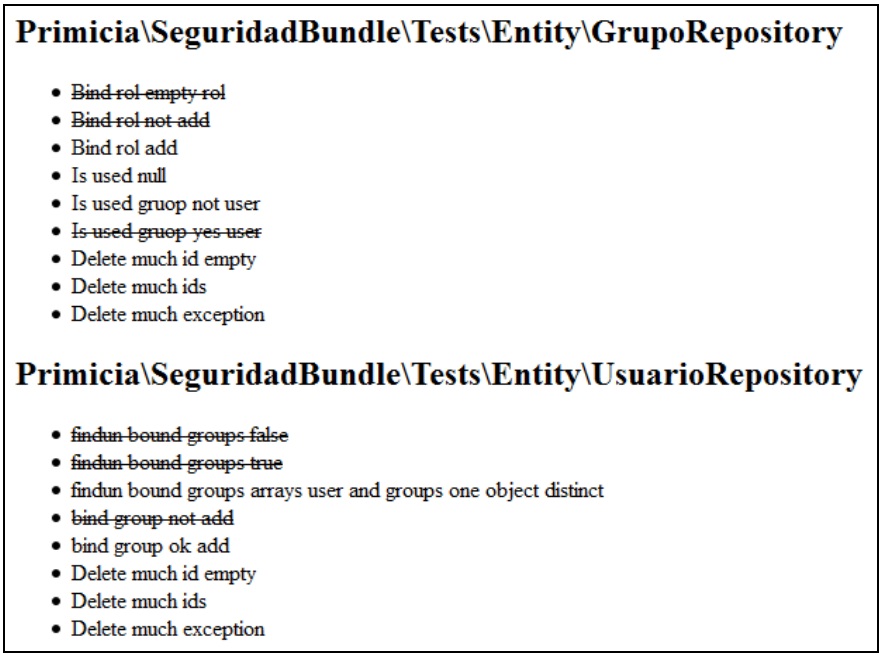
3.2.1 Registro del estado de las pruebas unitarias

Se registraron la cantidad de errores detectados por las pruebas, ubicándolos en las funcionalidades especificadas a probar. Se actualiza el plan general luego del estudio realizado de cada error detectado.

3.2.2 Registro del estado de los módulos

Se generan diferentes reportes debido a las funcionalidades y ventajas que aporta el framework seleccionado las cuales brindan información de la ejecución y fallos detectados. A continuación se muestran ejemplificaciones de informes generados.

La Figura 17 muestra un reporte en formato HTML de la ejecución de la colección de pruebas implementadas mediante las opciones de línea de comandos[9]. Se identifica de esta manera el estado (subrayado representa fallo) de cada prueba por clases del sistema bajo pruebas.



Primicia\SeguridadBundle\Tests\Entity\GrupoRepository

- ~~Bind rol empty rol~~
- ~~Bind rol not add~~
- Bind rol add
- Is used null
- Is used gruop not user
- ~~Is used gruop yes user~~
- Delete much id empty
- Delete much ids
- Delete much exception

Primicia\SeguridadBundle\Tests\Entity\UsuarioRepository

- ~~findun bound groups false~~
- ~~findun bound groups true~~
- findun bound groups arrays user and groups one object distinct
- ~~bind group not add~~
- bind group ok add
- Delete much id empty
- Delete much ids
- Delete much exception

Figura 17: Reporte de errores de PHPUnit 4.5.0 en formato HTML.

La Figura 18 muestra un reporte en formato texto de la ejecución de la colección de pruebas implementadas mediante las opciones de línea de comandos. Se identifica de esta manera el estado (la letra “x” representa fallo) de cada prueba por clases del sistema bajo pruebas. Además se generan reporte de cobertura de código en diferentes formatos de gran repercusión y ayuda para el análisis del código probado.

[9] phpunit --testdox-html <ubicación del archivo donde se guerdará>.

```
Primicia\MediasBundle\Tests\Entity\AutorRepository
 [ ] Delete much empty ids
 [x] Delete much ids
 [x] Delete much exception

Primicia\MediasBundle\Tests\Entity\Autor
 [x] Get id null
 [x] Get nombre
 [x] Set nombre leters
 [ ] Set nombre number
 [x] Get create at
 [x] Setcreate at
 [x] Get update at
 [x] Set update at
 [ ] Set create at after update at
 [x] test  toString
 [ ] test  toStringNumbers
```

Figura 18: Reporte de errores de PHPUnit 4.5.0 en formato de texto.

A continuación se muestra, en las Figura 19 y 20 el reporte de cobertura de código generado, donde se muestran en porcientos: el valor de líneas de código ejecutables que fueron ejecutadas (*Lines*), el valor de funciones y métodos invocados (*Functions and Methods*) y el valor de las clases y atributos que fueron ejecutadas.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		13.61%	112 / 823		30.99%	22 / 71		42.11%	8 / 19
Controller		0.00%	0 / 522		0.00%	0 / 37		0.00%	0 / 3
DependencyInjection		100.00%	8 / 8		100.00%	2 / 2		100.00%	2 / 2
Entity		70.83%	68 / 96		63.64%	7 / 11		50.00%	2 / 4
Form		100.00%	16 / 16		100.00%	6 / 6		100.00%	2 / 2
Handler		100.00%	10 / 10		100.00%	4 / 4		100.00%	1 / 1
Listeners		100.00%	10 / 10		100.00%	3 / 3		100.00%	1 / 1
Menu		0.00%	0 / 158		0.00%	0 / 7		0.00%	0 / 5
SeguridadBundle.php		0.00%	0 / 3		0.00%	0 / 1		0.00%	0 / 1

Figura 19: Reporte de análisis de cobertura de código generado en PHPUnit 4.5.0. Módulo seguridad.

	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total		28.83%	647 / 2244		62.18%	194 / 312		55.74%	34 / 61
Adapter		100.00%	10 / 10		100.00%	3 / 3		100.00%	1 / 1
Annotation		81.16%	56 / 69		91.30%	21 / 23		75.00%	3 / 4
Controller		0.00%	0 / 1337		0.00%	0 / 90		0.00%	0 / 11
DataTransformer		87.23%	41 / 47		83.33%	5 / 6		50.00%	1 / 2
DependencyInjection		64.71%	22 / 34		50.00%	1 / 2		50.00%	1 / 2
Driver		100.00%	20 / 20		100.00%	4 / 4		100.00%	1 / 1
Entity		74.91%	212 / 283		91.74%	100 / 109		66.67%	8 / 12
Exception								100.00%	1 / 1
Form		100.00%	195 / 195		100.00%	39 / 39		100.00%	13 / 13
Listener		100.00%	30 / 30		100.00%	8 / 8		100.00%	2 / 2
Menu		0.00%	0 / 103		0.00%	0 / 6		0.00%	0 / 6
Storage		52.59%	61 / 116		59.09%	13 / 22		40.00%	2 / 5
MediasBundle.php								100.00%	1 / 1

Figura 20: Reporte de análisis de cobertura de código generado en PHPUnit 4.5.0. Módulo gestión de medias.

3.2.3 Aseguramiento de la preservación de los productos para pruebas

Esta actividad es fundamental para el estudio y puesta en práctica de pruebas unitarias, por lo tanto se crea una copia de la colección de pruebas unitarias, las cuales pueden servir de ejemplo y sobre todo de guía para la implementación de nuevos casos de prueba. Además, se obtiene la documentación de todo el proceso ingenieril por el que se ha transitado.

3.3 Cierre del proceso

En el presente epígrafe se realiza un estudio de los resultados generados por el proceso de pruebas unitarias efectuado. Se muestra una serie de datos estadísticos de gran importancia cognoscitiva en el proceso de calidad de software, además de análisis tomándose datos de proyectos terminados e informes de tendencias del año 2014 del centro GEYSED.

3.2.4 Análisis sobre los casos de pruebas que fallaron. Módulo seguridad

La Figura 21 muestra los datos representativos de las pruebas realizadas al módulo seguridad. Se establece una comparación entre la cantidad total de pruebas, las que tuvieron éxito y las que fallaron.

Para un total de 40 pruebas, 30 de ellas no detectaron ningún error, representando un 75% del total, mientras que las 10 restantes encontraron errores en la codificación del módulo, representando un 25%. Por lo tanto se puede decir que de cada cuatro pruebas realizadas, una constituye un fallo para el código bajo pruebas, representando índices elevados de riesgo para la calidad del sistema.

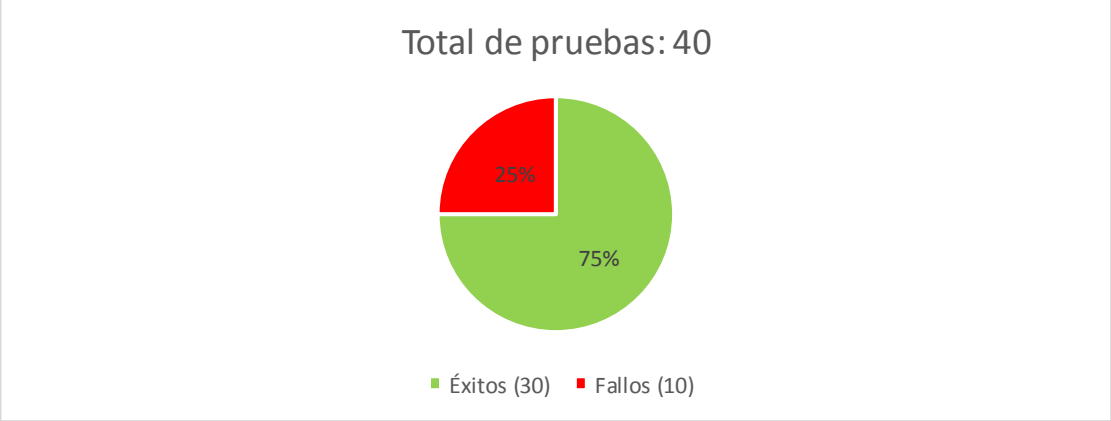


Figura 21: Gráfico representativo sobre las pruebas falladas. Módulo seguridad.

3.3.1 Análisis sobre las funcionalidades con errores. Módulo seguridad

La Figura 22 muestra los datos representativos de las funcionalidades del módulo seguridad. Se establece una comparación tomando como conjunto del dominio la cantidad total de funcionalidades del módulo, se comparan las funcionalidades con errores con las funcionalidades libres de errores.

Para un total de 14 funcionalidades, nueve de ellas no contienen errores representando un 64% del dominio, evidenciando así que cinco funcionalidades contienen errores en su codificación, representando el 36% restante. Se concluye que la cantidad de funcionalidades que contienen errores constituyen un porciento alto.

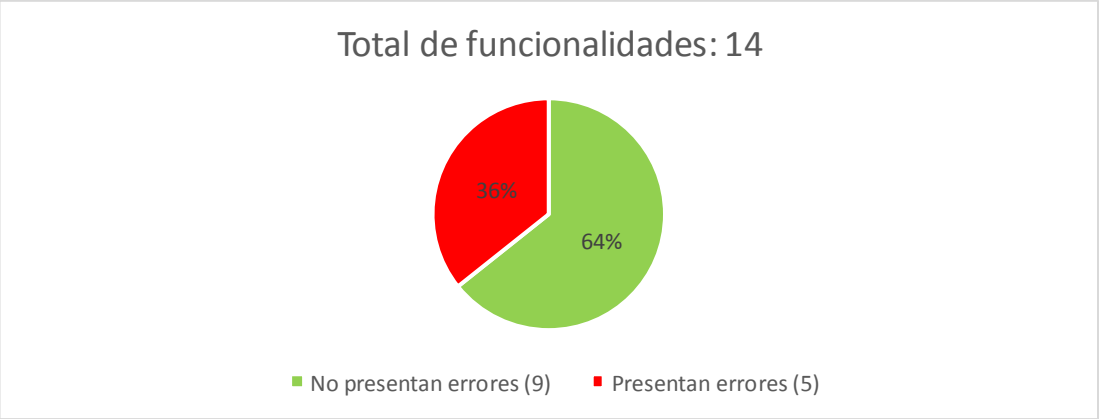


Figura 22: Gráfico representativo sobre las funcionalidades que presentan errores. Módulo seguridad.

3.3.2 Análisis sobre la distribución de los errores. Módulo seguridad

La Figura 23 muestra la relación establecida entre las funcionalidades pertenecientes al módulo seguridad, clasificadas según la prioridad para el cliente, aspecto fundamental a tener en cuenta para la calidad del producto. De dicha clasificación (alta, media y baja) se muestra la cantidad de funcionalidades que presentan errores y la cantidad de los mismos.

Se puede afirmar que las funcionalidades más afectadas son las de prioridad media y baja, la primera de estas tiene solo una funcionalidad con error, representando un 12,5%. Las funcionalidades de baja prioridad son las más afectadas, las mismas presentan el total, o sea el 100%.

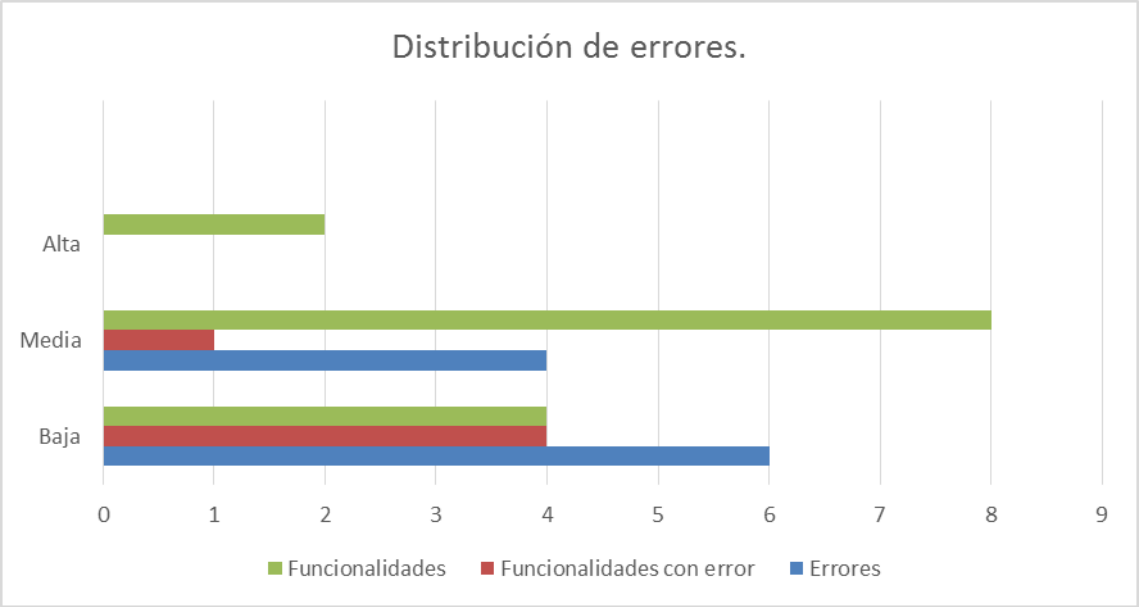


Figura 23: Gráfico sobre la distribución de errores. Módulo seguridad.

3.3.3 Análisis sobre los casos de pruebas que fallaron. Módulo gestión de medias

La Figura 24 muestra los datos representativos de las pruebas realizadas al módulo gestión de medias. Se establece una comparación entre la cantidad total de pruebas, las que tuvieron éxito y las que fallaron.

Para un total de 257 pruebas, 220 de ellas no detectaron ningún error, representando un 86% del total, mientras que las 37 pruebas restantes encontraron errores en la codificación del módulo, representando un 14%. Se afirma que cada siete pruebas realizadas aproximadamente una constituye un fallo para el código bajo pruebas, representando índices medios de riesgo para la calidad del sistema.

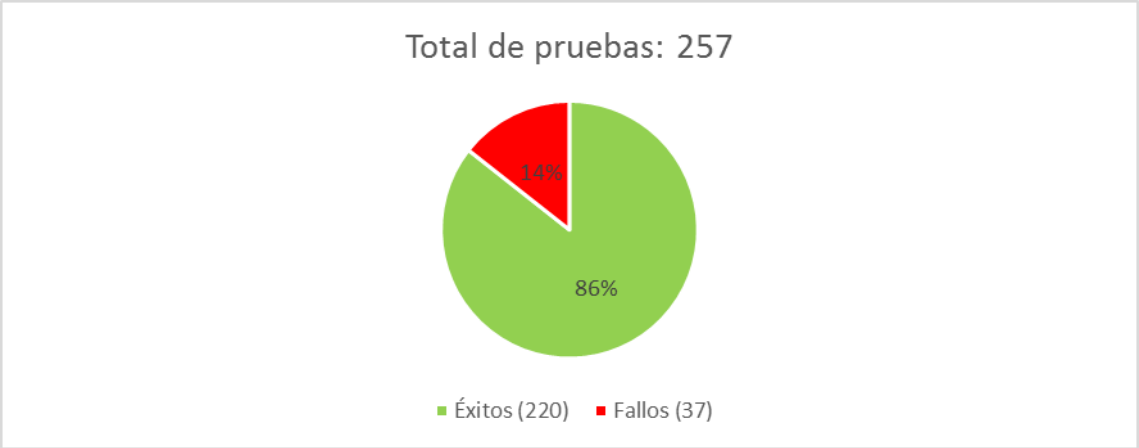


Figura 24: Gráfico representativo sobre las pruebas falladas. Módulo gestión de medias.

3.3.4 Análisis sobre las funcionalidades con errores. Módulo gestión de medias

La Figura 25 muestra los datos representativos de las funcionalidades del módulo gestión de medias. Se comparan las funcionalidades con errores con las funcionalidades libres de errores.

Para un total de 24 funcionalidades, 12 de ellas no contienen errores, representando un 50% del dominio, evidenciando así que la misma cantidad de funcionalidades contienen errores en su codificación, representando el 50% restante. Por lo tanto se infiere que la cantidad de funcionalidades que contienen errores constituyen un porciento alto.

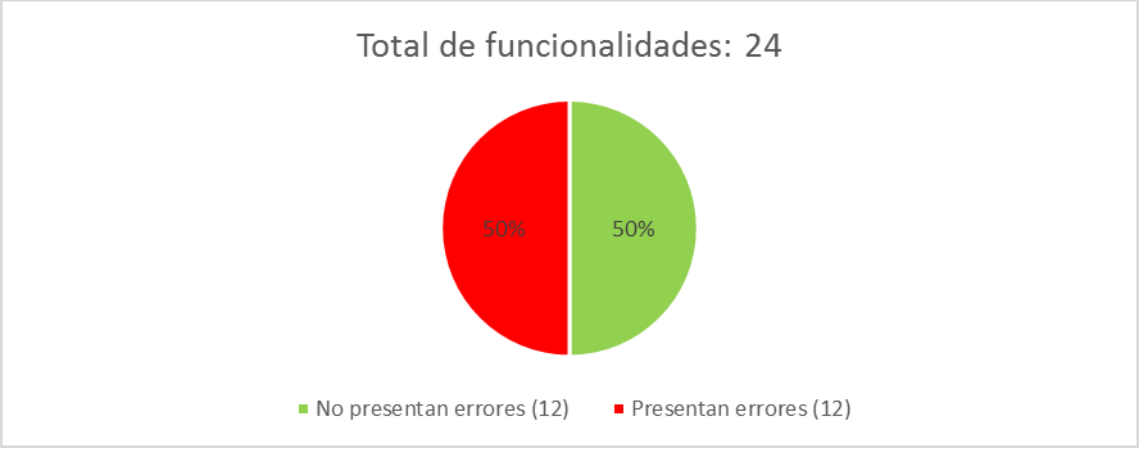


Figura 25: Gráfico representativo sobre las funcionalidades que presentan errores. Módulo gestión de medias.

3.3.5 Análisis sobre la distribución de los errores. Módulo gestión de medias

La Figura 26 muestra la relación establecida entre las funcionalidades pertenecientes al módulo gestión de medias, clasificadas según la prioridad para el cliente, aspecto fundamental a tener en cuenta para la calidad del producto. De esta clasificación (alta, media y baja) se muestra la cantidad de funcionalidades que presentan errores y la cantidad de los mismos, destacando en particular los errores que afectan funcionalidades de diferentes clasificaciones (Errores en común).

Se puede afirmar que las funcionalidades más afectadas son las de prioridad alta y media, la primera de estas tiene seis funcionalidades con error representando un 100%. Son afectadas seis funcionalidades de media prioridad, para un 75%.

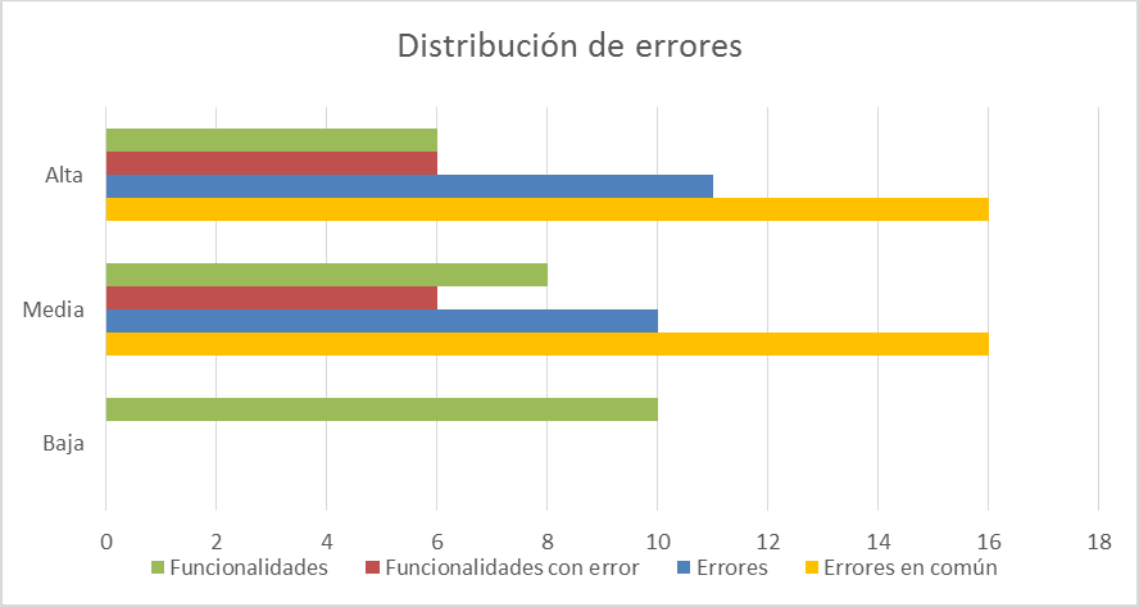


Figura 26: Gráfico sobre la distribución de errores. Módulo gestión de medias.

3.3.6 Análisis de comportamiento usando proyectos terminados del centro GEYSED

Para establecer el análisis siguiente se tomaron los datos referentes a los informes de errores (documentos de no conformidades) de los proyectos Sistema para la automatización del proceso de producción de series y telenovelas (CPT), Mesa Redonda (MR), Sistemas de información geográfica para el grupo empresarial de la industria alimentaria (SIGGEIA) y Sistema de gestión audiovisual para el centro

de información del comité central del PCC (SGAOICCPCC) pertenecientes al centro GEYSED y además en fase de terminados.

Estos datos fueron filtrados donde solo se muestran los errores que representan un subdominio válido (errores de validación, funcionalidad y excepciones) para la investigación, obviando así otros tipos de errores que no se detectan en las pruebas unitarias.

La Figura 27 muestra los valores de errores de los cuatro proyectos terminados mencionados anteriormente, además su distribución de comportamiento, la representación del promedio (39.5), además la representación de los errores de cada uno de los módulos bajo pruebas y el total de los mismos.

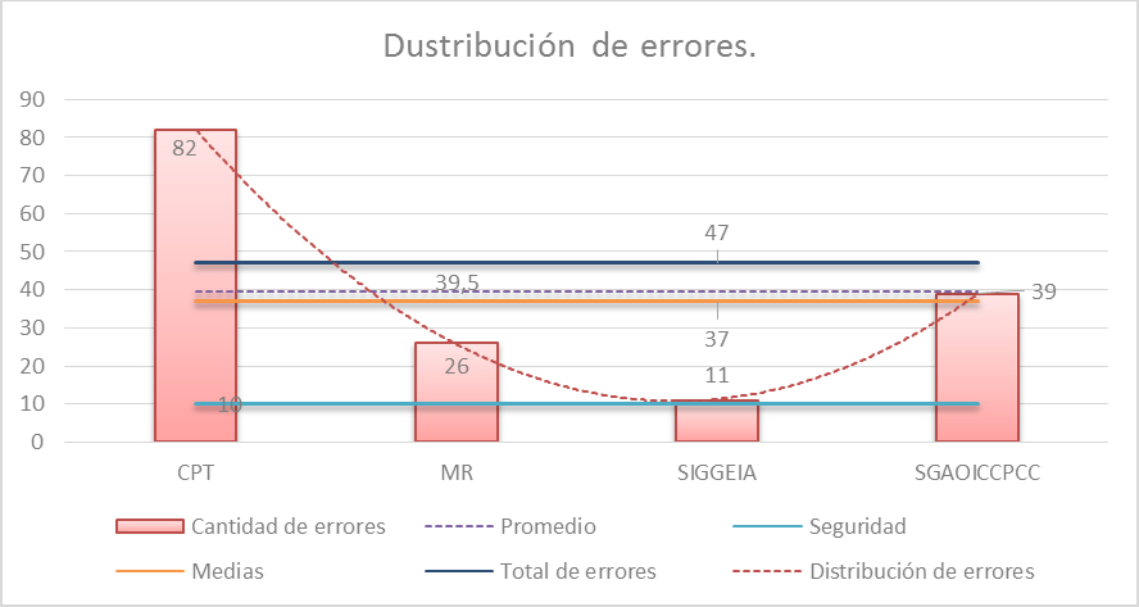


Figura 27: Distribución de errores de proyectos terminados del centro GEYSED.

Puede afirmarse que el promedio de errores de los módulos bajo pruebas de XILEMA PRIMICIA 2.0 es de 23,5 por debajo de la media establecida en el gráfico anterior. Sin embargo estos dos módulos solo constituyen el 28% del sistema, de procederse a calcular el valor estimado de errores de la plataforma a través del promedio se tiene que sería aproximadamente 165 errores superando con creces la media establecida.

Analizándose el módulo gestión de medias individualmente, se evidencia que la cantidad de errores (37) está bastante cerca de la media, no así en el caso del módulo seguridad (10). Es válido aclarar que el último módulo es el de menor complejidad del sistema.

3.3.7 Análisis de comportamiento usando datos del año 2014

Para establecer el análisis de comportamiento se tomaron los datos referentes a los informes de tendencias del Centro GEYSED por trimestres, durante el año 2014. Estos datos fueron filtrados y solo se muestran los errores que representan un subdominio válido (errores de validación, funcionalidad y excepciones) para la investigación, obviando así otros tipos de errores que no se detectan en las pruebas unitarias.

La figura 33 muestra los valores de errores correspondientes a los informes de tendencias pertenecientes a los trimestres del año 2014, realizando de esta manera la distribución de comportamiento a lo largo del año, la representación del promedio (56), además la representación de los errores de cada uno de los módulos bajo pruebas y el total de los mismos.

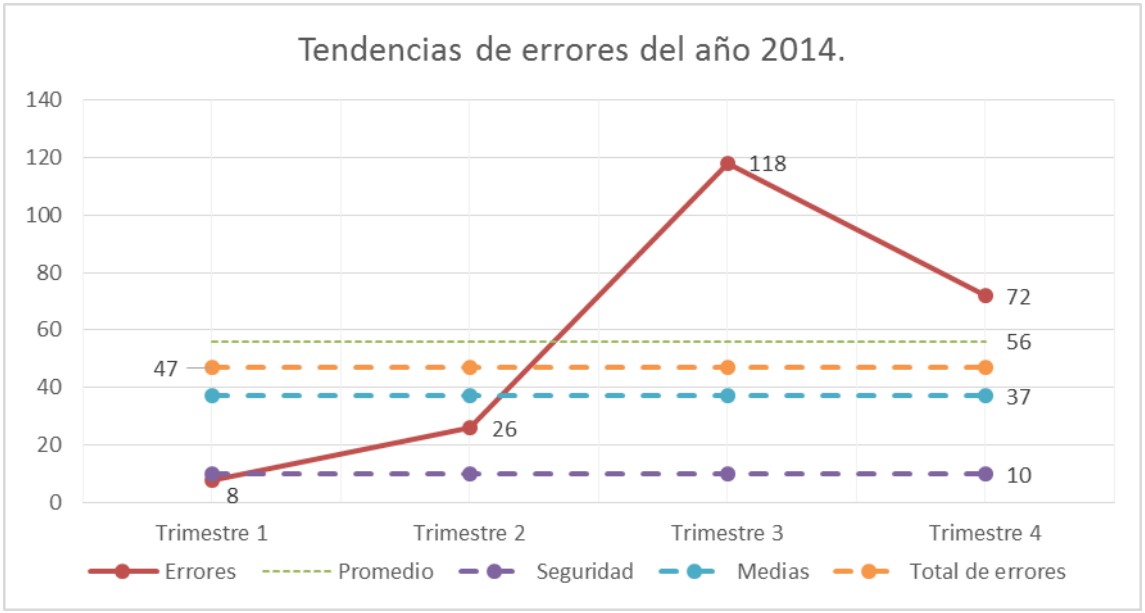


Figura 28: Distribución de errores del centro GEYSED por trimestres del año 2014.

El promedio de errores (23,5) pertenecientes a los módulos seguridad y gestión de medias del sistema XILEMA PRIMICIA 2.0 se encuentra por debajo de la media establecida en el gráfico anterior. Sin

embargo estos dos módulos solo constituyen el 28% del sistema por lo que si se procede a calcular el valor estimado de errores de la plataforma a través del promedio se tiene que sería aproximadamente 165 errores superando con creces la media establecida.

Analizándose el módulo gestión de medias individualmente, se tiene que la cantidad de errores (37) del mismo está cerca de la media, no así en el caso del módulo seguridad (10), es válido aclarar que el último módulo es el de menor complejidad del sistema.

3.4 Conclusiones parciales

Luego de establecer una serie de análisis referentes a los datos de los módulos seguridad y gestión de medias con los datos del Centro GEYSED se concluye lo siguiente:

- Los errores de los módulos que presentan su código bajo pruebas (47) afectan en gran medida las funcionalidades (38) descritas para el sistema. Se determinan 17 funcionalidades defectuosas para un 44.7% del total probadas.
- Luego de realizarse los análisis de comportamiento se puede afirmar que la puesta en práctica del proceso de pruebas unitarias y la implementación de las mismas, reduce el tiempo de duración de las etapas de pruebas, ya que disminuye notablemente la cantidad de errores que se arrastran a etapas posteriores.

Conclusiones Generales

Con el cumplimiento de los objetivos y las tareas definidas para la investigación se logró obtener una solución a la problemática que originó la investigación actual. Una vez finalizada se concluye:

- La investigación posibilita la obtención de resultados satisfactorios de gran interés para las etapas de pruebas planificadas en los proyectos del Centro GEYSED, lo que brinda un aporte significativo para el proceso de la calidad.
- La documentación técnica generada durante el desarrollo de la investigación de acuerdo con el estándar IEEE Std. 1008-1987 y la utilización del framework para pruebas unitarias PHPUnit 4.5.0, permite comprender mejor la importancia del proceso de pruebas unitarias y su adaptación al proceso de desarrollo de software.
- Se garantiza la detección temprana de errores, lo que evita su propagación a etapas superiores y mayor calidad de productos de trabajo (lo que recursivamente repercute en el costo y tiempo de corrección).
- El software se obtiene con mayor robustez y entendimiento pues las pruebas unitarias inducen la refactorización.
- La aplicación del proceso de pruebas unitarias facilita que durante las pruebas funcionales y de integración se realicen menos iteraciones, pues no se arrastran errores que pueden ser detectados durante las pruebas unitarias, relativos a las unidades de software.

Recomendaciones

Cumplido el objetivo general de la investigación y teniendo en cuenta las experiencias obtenidas a lo largo de su desarrollo, se recomienda:

- Diseñar e implementar casos de pruebas unitarias a los módulos infocintas, editorial, señales, noticia y transmisión.
- Desarrollar pruebas funcionales automatizadas con el framework PHPUnit 4.5.0 y pruebas de aceptación a partir de su integración con Selenium para complementar el proceso pruebas en el Centro GEYSED.

Referencias

BAHIT, E. 2013. Hackers & Developers Magazine. , vol. 5.

BAKER, M. 2015. SimpleTest - Unit Testing for PHP. [en línea]. [Consulta: 21 marzo 2015]. Disponible en: http://www.simpletest.org/.

BERGMANN, S., 2015a. *PHPUnit Manual.* 2015. S.l.: Creative Commons Attribution 3.0.

BERGMANN, S. 2015b. PHPUnit – The PHP Testing Framework. *Welcome to PHPUnit!* [en línea]. [Consulta: 21 marzo 2015]. Disponible en: https://phpunit.de/.

CANÓS, J.H., LETELIER, P. y PENADÉS, M.C. 2011. Métodologías Ágiles en el Desarrollo de Software. . Valencia, España: DSIC -Universidad Politécnica de Valencia.,

CASTAÑEDA VARGAS, P.S. y GUEVARA SANDOVAL, L.A. 2010. MODELO DE PRODUCTIVIDAD BASADO EN COMPONENTES PARA LA FÁBRICA DE SOFTWARE. . Perú:

CEQUEA, M. y RODRÍGUEZ MONROY, C. 2012. Productividad y factores humanos. Un modelo con ecuaciones estructurales. , ISSN 0378-1844.

CROES, G. 2014. What is atoum? — atoum 1.0.0 documentation. [en línea]. [Consulta: 1 mayo 2015]. Disponible en: http://atoum-en.readthedocs.org/en/latest/index.html.

DEMING, W.E. 1989. *Calidad, productividad y competitividad: la salida de la crisis.* Madrid: Ediciones Diaz de Santos. SA. ISBN 84-87189-22-9.

EGUILUZ, J. 2012. *Desarrollo web ágil con Symfony 2.* S.l.: easybook 4.8-DEV.

FENTON, S. y JONES, M. 2011. Enhance Php. [en línea]. [Consulta: 1 mayo 2015]. Disponible en: http://enhancephp.com/.

FERNÁNDEZ SANZ, L. 2005. Un sondeo sobre la práctica actual de pruebas de software en españa. . Madrid, España: REICIS, Asociacion de Técnicos de Informática.

GUEST, S., GREENFIELD, J. y COOK, S. 2006. The architecture journal. ,

GUTIERREZ, D. 2012. Frameworks y Componentes. . Venezuela.: Universidad de los Andes.,

GUTIÉRREZ, J.J. 2012. ¿Qué es un framework web? . S.l.:

HERNÁNDEZ MERAZ, E. 2009. Factores e indicadores relacionados con la productividad de una empresa dedicada al desarrollo de software. . México:

IBM, C. 2006. Classic RUP for SOMA.es. [en línea]. [Consulta: 5 mayo 2015]. Disponible en: http://cgrw01.cgr.go.cr/rup/RUP.es/LargeProjects/index.htm#core.base_rup/guidances/supportingmaterials/welcome_2BC5187F.html.

IEEE, I. of E. and E.E., 1990. *Standard Glossary of Software Engineering Terminology*. 1990. S.l.: s.n.

IEEE, I. of E. and E.E., 1993. *IEEE Standard for Software Unit Testing.* 1993. S.l.: The Institute of Electrical and Electronics Engineers.

IEEE, I. of E. and E.E. 2004. Cuerpo del conocimiento de la Ingeniería de Software. Capítulo 11 - Calidad del software. *GUÍA AL CUERPO DE CONOCIMIENTO DE LA INGENIERÍA DEL SOFTWARE. SWEBOK*. S.l.: s.n., ISBN 0-7695-2330-7.

IEEE, I. of E. and E.E. 2014. *Guide to the Software Engineering. SWEBOK®.* S.l.: Pierre Bourque, École de technologie supérieure (ÉTS), Richard E. (Dick) Fairley, Software and Systems Engineering Associates (S2EA).

INTECO, L.N. de C. del S. de 2009. INGENIERÍA DEL SOFTWARE: METODOLOGÍAS Y CICLOS DE VIDA. . España:

INTI ARGENTINA, P.S. 2005. Calidad de Software: Modelos y estándares. Software Mejora de la Eficiencia y la Competitividad de la Economía Argentina. . Argentina.

ISO, I.O. for S., 2000. *Norma ISO 9000:2000*. 2000. S.l.: s.n.

JENKINS, M. 2004. Diseño e Implementación de un Sistema de Administración de la Calidad del Software para una Institución Financiera. *SISTEMAS, CIBERNÉTICA E INFORMÁTICA.* 1. Costa Rica: s.n., ISBN 1690-8627.

JOVANOVIĆ, I. 2006. Software Testing Methods and Techniques. . S.l.:

JOYANES AGUILAR, L. 2007. IV SIMPOSIO INTERNACIONAL DE SISTEMAS DE INFORMACIÓN E INGENIERÍA DE SOFTWARE EN LA SOCIEDAD DEL CONOCIMIENTO. *SISOFT2006*. Cartagena de Indias, Colombia: s.n.,

KHALED, M.M., RAFA, E.A.-Q. y MOHAMMAD, I.M. 2009. Classification of Software Testing Tools. *Second International Conference on Computer and Electrical Engineering.* S.l.: s.n.,

KONKA, M. of S.T. in S.E. and M.B.B. 2011. A case study on Software Testing. . Göteborg, Sweden.: VOLVO TECHNOLOGY.

LÉPINE, J.-F. 2012. Behat Documentation — Behat 2.5.3 documentation. [en línea]. [Consulta: 21 marzo 2015]. Disponible en: http://docs.behat.org/en/v2.5/.

LETELIER, P. 2006. Proceso de desarrollo de software. . S.l.:

LÓPEZ CARRILLO, I., 2013. *Cómo mejorar la calidad del software a través de una gestión adecuada de la productividad de las pruebas* [en línea]. 2013. S.l.: Leda-mc. Disponible en: http://www.leda-mc.com/.

MERAYO CASTELLANO, I. 2014. Testing en PHP. . S.l.: Agilecyl,

NETBEANS TEAM, Weycog. 2015. Welcome to NetBeans. [en línea]. [Consulta: 21 marzo 2015]. Disponible en: https://netbeans.org/.

PMBOOK, P.M.I. 2009. *Guía de los Fundamentos para la Dirección de Proyectos (Guía del PMBOK). Cuarta Edición*. Project Management Institute, Inc. S.l.: s.n. ISBN 978-1-933890-72-2.

POTENCIER, F. y WEAVER, R. 2013. *SYMFONY 2. El libro oficial.* S.l.: s.n.

POTENCIER, F. y ZANINOTTO, F. 2012. *SYMFONY. La guía definitiva.* S.l.: s.n.

PRESSMAN, R.S. 2002. *Ingeniería del Software. Un enfoque práctico*. S.l.: s.n.

PRIMICIA, U. de las C.I., 2012a. *Especificación de casos de uso. Proyecto PRIMICIA. Módulo Medias*. 2012. S.l.: s.n.

PRIMICIA, U. de las C.I., 2012b. *Especificación de casos de uso. Proyecto PRIMICIA. Módulo Seguridad.* 2012. S.l.: s.n.

PRIMICIA, U. de las C.I., 2013. *Especificación de requisitos de software. Proyecto PRIMICIA.* 2013. S.l.: s.n.

TERRENCE, P. y ZELKOWITZ, M. 2000. *LENGUAJES DE PROGRAMACION: DISEÑO E IMPLEMENTACION.* 3. Mexico: s.n. 3577. ISBN 0-13-678012-1.

UCI, U. de las C.I. 2012. Entorno Productivo | Portal de la Universidad de las Ciencias Informáticas. [en línea]. [Consulta: 26 mayo 2015]. Disponible en: http://www.uci.cu/?q=entorno-productivo.

VALBUENA APONTE, Á.M. 2014. GUÍA COMPARATIVA DE FRAMEWORKS PARA LOS LENGUAJES HTML 5, CSS Y JAVASCRIPT PARA EL DESARROLLO DE APLICACIONES WEB. . Pereira: UNIVERSIDAD TECNOLÓGICA DE PEREIRA.