Universidad de las Ciencias Informáticas

Facultad 1

Título: Propuesta de procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autoras:

Marlen Thaureaux Martínez y Dania Rosa Hernández Quintana

Tutora: Ing. Geidis Sánchez Michel

Junio, 2009

"Si puedes creer, al que cree todo le es posible."

Jesús de Nazaret

DECLARACIÓN DE AUTORÍA.

Declaramos que somos los únicos autores del trabajo titulado: “Propuesta de procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2” y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

Dania Rosa Hernández Quintana

_____

Firma del Autor

Marlen Thaureaux Martínez

_____

Firma del Autor

Geidis Sánchez Michel

_____

Firma del Tutor

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título: Propuesta de procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2.

Autores: Dania Rosa Hernández Quintana

Marlen Thaureaux Martínez

Tutor: Ing. Geidis Sánchez Michel

El documento tiene buena presentación, está legible y posee una correcta organización interna mostrando una alta coherencia en el desempeño de la problemática planteada por las tesistas.

El tutor del presente Trabajo de Diploma considera que durante la ejecución de esta tesis las tesistas mostraron un alto dominio del tema desarrollado, describiendo correctamente la necesidad de mejorar el procedimiento de pruebas de calidad de la metodología MA-GMPR-UR2.

La propuesta de solución introduce mejoras en el proceso de prueba de calidad para los proyectos que utilicen dicha metodología, permitiendo revelar las causas de nuevos errores, insuficiencias de funcionalidad o discrepancias funcionales con respecto al comportamiento que se esperaba del software. Partiendo de la generación de casos de pruebas, la propuesta brinda la posibilidad de documentar las pruebas realizadas al software, siendo más fácil para el equipo de soporte encontrar las causas de errores futuros en el producto.

Por todo lo anteriormente expresado considero que las estudiantes están aptas para ejercer como Ingenieras en Ciencias Informáticas; y propongo que se le otorgue al Trabajo de Diploma la calificación de ____ puntos.

_____ _____

Firma Fecha

DATOS DE CONTACTO

Marlen Thaureaux Martínez,

Correo: mthaureaux@estudiantes.uci.cu .

Ciudad de la Habana, Cuba

Dania Rosa Hernández Quintana

Correo: drhernandez@estudiantes.uci.cu.

Ciudad de la Habana, Cuba

Ing. Geidis Sánchez Michel

Ingeniera en Ciencias Informáticas.

Correo: gsanchez@uci.cu , teléfono 835-8863

Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

AGRADECIMIENTOS

Me gustaría agradecer primero a Dios por haber permitido graduarme. Por haberme sacado del mundial de P1, de Matemática 1, por librarme de muchas otras asignaturas como P4, la prueba de nivel de Programación, por el milagro de la prueba de Ingles. Porque un día puso en mi corazón el deseo de buscarle. A mis padres que me apoyaron para nunca darme por vencida, a mi abuela, a mi tío y mi familia completa por estar ahí. A mis hermanos de la fe el regalo que Dios me dio en este lugar y lo que más le agradezco porque era todo lo que yo esperaba y necesitaba. A mis amigos Massiel, Irán, Yaimí. A mis compañeros de aula, a los que quedaron atrás y no se graduaron. A mis compañeras de cuarto, Ailyn, Mireidys y Noralys. A mi tutora, a mi compañera de tesis… las quiero mucho. A los que no saben si se graduarán, nada es imposible para Dios. ¡Si se puede!

Dania Rosa Hernandez Quintana

Cuando la gratitud es tan grande las palabras sobran y precisamente en este momento las palabras se me agotan para agradecer a tantas personas que han tenido que ver con que suceda este momento tan especial en mi vida. Quiero agradecerle a mi abuelita que siempre esta conmigo, apoyándome y ayudándome a salir adelante. A mi mama, por ser la mejor del mundo, que ha dedicado todo para que sus hijas sean cada vez mejor y sin ella estos 5 años tan importantes de mi vida no hubiesen sido posibles. A mi papa, por ser tan magnífico, por ser mi ejemplo, se que soy su orgullo y este camino largo por el que he transitado y que no ha sido fácil, he llegado al final porque ha sabido guiarme. A mi hermana que siempre esta conmigo, es mi amiga y quiero que sepas que estoy muy orgullosa de ti. A mi familia que de una forma u otra contribuyeron a mi formación profesional. A mis amigos, que son muy importantes para mi vida, porque son amigos de verdad, Dai, Kleidys, Yoa, Mario, Oli, Daliana, Kirenia, Mary, Yaima, Ismel, la nueva Yuliet. A mi compañera de tesis que ha sabido soportarme este año. A mi tutora, por dedicarnos su tiempo en apoyarnos y ayudarnos siempre que la necesitamos. De manera especial a la Revolución, por darme la oportunidad de convertirme en una mujer preparada. A TODOS!!! Para que nadie se quede afuera.

Marlen Thaureaux Martínez

DEDICATORIA

Este trabajo esta dedicado a mi mamá que tanto se esforzó
porque yo llegase hasta aquí, a mi papá que es el mejor regalo que me han dado,
a mi abuela que siempre estuvo apoyándome y alentándome.
A mis hermanos Hilario, Martica y el batallón de hermanos
de la fe que siempre me acompañan.
Y sobre todas las cosas a mi Dios Jesús de Nazaret por ser mi inspiración,
la motivación perfecta que llegó a tiempo y que nunca me faltó
aún en los momentos más difíciles y que me permitió llegar al final.

Dania Rosa Hernández Quintana.

A mi abuelita que siempre esta conmigo,
a mis padres por darme lo mejor que me pudieron dar:
Educación, Valores, Regaños y todo su Amor,
a mi hermana para que su tesis también me la dedique.
a mi familia, a todos mis amigos que son mi otra familia.
A todos, muchas gracias.

Marlen Thaureaux Martínez

RESUMEN

Metodologías ágiles como Scrum y XP son muy útiles para mejorar la calidad y reducir las desviaciones en los proyectos que son aplicadas. Como parte indispensable para lograr un software con la calidad que requiere el cliente es indispensable el desarrollo y ejecución de pruebas. En el año 2008 surge en la Universidad de las Ciencias Informáticas una propuesta de metodología llamada MA-GMPR-UR2 la cual esta basada en metodologías ágiles de punta como Scrum y XP. Las pruebas son un elemento esencial dentro de las mencionadas metodologías ágiles donde juegan un papel director debido al cambio constante que pueden efectuarse en los requisitos funcionales. El presente trabajo profundiza en la fase de pruebas y propone un procedimiento de pruebas de calidad aplicable a la metodología MA-GMPR-UR2. El procedimiento que propone este estudio favorece el incremento de las pruebas de calidad en cada una de las cortas iteraciones de la metodología, minimiza lo más posible el tiempo en que se desarrollen las pruebas y registra la documentación con los resultados de cada una de las pruebas realizadas al producto para garantizar la calidad en cada una de las pequeñas entregas del programa que se le realizan al cliente y en el producto final.

Palabras Claves: Metodologías ágiles, calidad, pruebas de calidad, procedimiento de pruebas.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

INTRODUCCIÓN

Asegurar la calidad de un producto de software se ha convertido en una necesidad en el mundo de la computación actual. En el software, la calidad, no es opcional, no puedes elegir elaborar software de menor o mayor calidad y rebajar o elevar el precio del mismo. El mercado simplemente aplastaría al productor. En un mundo donde la competencia es la base del éxito no se puede simplemente jugar con la calidad de lo que se produce.

Un software de calidad reduce los costes de desarrollo y acorta los plazos. Mientras más tarde se descubren los errores más costosos resultan. Es importante conocer el hecho de que los usuarios cada vez son menos flexibles con los errores en las aplicaciones. Nadie compra un software que tiene mala calidad.

La manera más rápida de desarrollar un software es hacerlo con la calidad requerida desde el principio. Como parte de esa idea las metodologías ágiles plantean que se puede desarrollar un software que cumpla con las especificaciones de los clientes desde el mismo inicio y que el cambio continúo en los requisitos no tiene por que afectar la calidad del software siempre y cuando se siga un procedimiento eficiente.

Actualmente, en la Universidad de las Ciencias Informáticas (UCI) se trabaja con la metodología MA-GMPR-UR2[1], donde se fusionan dos metodologías ágiles en la actualidad: Scrum y XP. Esta metodología fue propuesta realizada en junio de 2008 por la ingeniera Gladys Marsi Peñalver Romero para ser utilizada en los proyectos productivos de la facultad 10.

En estos proyectos las pruebas que se le realizan al software son las pruebas de aceptación por el cliente que trabaja con el equipo de desarrollo y las pruebas unitarias realizadas por los desarrolladores. Actualmente estas pruebas se realizan manuales y no cuentan con un procedimiento establecido que permita tener una idea de cómo se llevó a cabo el proceso de

[1] Metodología Ágil- Gladys Marsi Peñalver Romero-Revisión 2

pruebas y poder detectar cuáles fueron las principales deficiencias que afectaron la calidad del software para proceder a su mitigación.

De aquí que el **problema científico** sea ¿cómo garantizar la calidad del proceso de pruebas de software utilizando la metodología MA-GMPR-UR2? Siendo el **objeto de estudio** la calidad de software. Como **objetivo general** se plantea elaborar un procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2. El **campo de acción** que abarca la investigación son las pruebas de calidad de software para la metodología MA-GMPR-UR2.

Como **hipótesis** se plantea que si se lleva a cabo un procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2 entonces se mejorará la calidad del proceso de pruebas de calidad de software de los proyectos que trabajen con dicha metodología.

Variables:

Variable independiente: procedimiento de pruebas de calidad.

Variable dependiente: proceso de pruebas de calidad de software

Tabla 1. Operacionalización de las variables

Variables	Dimensión	Indicador	Sub-indicadores	Unidad de Medida
Procedimiento de pruebas de calidad	Proyecto que utilizan la metodología MA-GMPR-UR2	Satisfacción a las necesidades de la metodología MA-GMPR-UR2.	Muy bajo	1
			Bajo	2
			Regular	3
			Bien	4
			Excelente	5
		Posibilidad de aplicación.	Muy bajo	1
			Bajo	2
			Regular	3
			Bien	4

			Excelente	5
		Adaptabilidad a los proyectos productivos.	Muy bajo	1
			Bajo	2
			Regular	3
			Bien	4
			Excelente	5
		Mejora del proceso de prueba	Muy bajo	1
			Bajo	2
			Regular	3
			Bien	4
			Excelente	5

Objetivos específicos:

- Realizar el marco teórico de la investigación.
- Definir un procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2.
- Evaluar el procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2.

Para cumplir los objetivos se plantean las siguientes **tareas de la investigación:**

- Estudiar la bibliografía para caracterizar el estado actual de los procedimientos de pruebas de calidad.
- Realizar un análisis de las diferentes fases según la metodología de desarrollo MA-GMPR-UR2 y los artefactos generados.
- Desarrollar un análisis del proceso de pruebas en todas las etapas.
- Desarrollar la técnica de panel de experto para validar la solución propuesta

- Interpretar los resultados obtenidos.

Modelo metodológico

Población: 12 proyectos.

Unidad de estudio: procedimiento de pruebas de calidad de la metodología MA-GMPR-UR2

Muestra: 2 proyectos.

Tamaño: Son 2 proyectos los que se seleccionan a la hora de realizar las pruebas lo cual representa el 16.6% de la población.

Criterio de selección: Se utiliza el muestreo intencional dentro de las técnicas no probabilísticas. Los proyectos donde se estudiará como se desarrolla el proceso de pruebas de calidad utilizando la metodología MA-GMPR-UR2 son: ServiceDesk (Sistema de Gestión de Reportes) y dotproject SXP (módulo encargado de analizar los reportes en formato XML que llegan al servidor).

Estrategia investigativa:

Descriptiva: Con el uso de esta estrategia de investigación se logrará describir el procedimiento de pruebas de calidad y reflejar lo esencial y más significativo del mismo.

Métodos científicos de investigación empleados:

Métodos teóricos:

Analítico-sintético: Este método permite comprender cómo se realizan las pruebas de calidad en proyectos donde se aplican metodologías ágiles así como realizar un análisis profundo de todos los procedimientos, sintetizar los conocimientos, y de esta forma descubrir todas las características generales. Mediante el estudio de las pruebas factibles para este tipo de

procedimientos se documenta en el presente trabajo los principales objetivos, logros, o deficiencias que se derivan de cada prueba.

Histórico- lógico: Este método permite estudiar las pruebas de calidad que propone la metodología así como las pruebas que se realizan en otros proyectos, para así plantear un procedimiento de pruebas de calidad que garantice la calidad en el desarrollo del proyecto.

Métodos empíricos:

Encuesta: A la hora de realizar la validación de la propuesta mediante la técnica de panel de experto, se usará la encuesta para obtener las opiniones de experto en correspondencia con la propuesta de solución.

El presente trabajo de diploma consta de 3 capítulos:

Capítulo 1. Fundamentación teórica: Se realiza la fundamentación teórica de la investigación, abordando los principales conceptos de calidad, pruebas de calidad, tipos de pruebas de calidad de software, metodologías ágiles como XP, Scrum y MA-GMPR-UR2.

Capítulo 2. Propuesta de solución: Se realiza la descripción de la solución propuesta, elaborando un procedimiento de pruebas de calidad para la metodología MA-GMPR-UR2.

Capitulo 3. Validación de la propuesta: Se realiza la evaluación de la propuesta de solución, utilizando la técnica de panel de experto.

CAPÍTULO I. FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En el presente capítulo se realiza el estudio del estado del arte donde se brinda una visión general de los aspectos relacionados con calidad de software, las pruebas de calidad, una breve descripción de las metodologías ágiles Scrum y XP y una explicación de la metodología MA-GMPR-UR2.

1.2. Calidad de software

Actualmente el desarrollo del software se ha incrementado, gran parte del mismo ha sido realizado de forma desorganizada y poco documentada. Considerando el aumento exponencial que sufrirá en los próximos años, surge la necesidad de lograr una evaluación sistema del proceso de desarrollo de software que garantice un producto final con calidad.

En 1996 el presidente de la Academia Internacional de Calidad define la calidad como: “Un sistema eficaz para integrar los esfuerzos de mejora de la gestión de los distintos grupos de la organización para proporcionar productos y servicios a niveles que permitan la satisfacción del cliente” (1)

La Organización Internacional de Estándares (ISO) y la Comisión Internacional Electrotécnica (IEC) definen la calidad de software como: “la totalidad de características de un producto de software que le confiere la capacidad de satisfacer necesidades explícitas e implícitas”. (2)

La calidad del software Pressman la define como “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.” (3)

En resumen todas ellas coinciden en que la calidad de software es la capacidad que tiene el sistema de satisfacer las necesidades que el cliente demanda y superar las expectativas del mismo.

1.1.1 Proceso de pruebas de calidad

La calidad del software es medible y varía de un programa a otro, puede medirse después de elaborado el producto, pero esto puede resultar muy costoso, por lo que es imprescindible realizar pruebas de calidad durante todas las etapas del ciclo de vida del software.

Según el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) en 1991 define las pruebas de software como: "… una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente". (4)

Pressman define las pruebas de calidad de software como: "La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación". (5)

1.1.2 Tipos de pruebas de calidad de software

Existen varios niveles de prueba definidos y estos niveles a su vez especifican qué tipos y métodos de pruebas se deben utilizar en cada uno de ellos y cuáles son sus objetivos. A continuación se hace referencia a los niveles de pruebas:

- **Unitarias**

    1. Enfocadas al código fuente de los componentes.
    2. Para verificar todos los flujos de control.
    3. Primero pasa por la revisión del programador.

La práctica que se utiliza para desarrollar las pruebas unitarias es Desarrollo Dirigido por Pruebas (TDD): es una técnica de programación que plantea escribir primero los casos de prueba y luego implementar el necesario para ejecutarla.

Los mayores beneficios de TDD son: (6)

- Antes de escribir cualquier fragmento de código, se debe escribir las pruebas automatizadas para comprobar la funcionalidad de ese futuro código. Como el código no existe todavía, inicialmente la prueba falla.

- Una vez comenzadas las pruebas, se debe eliminar el código duplicado.

- **De integración**

    1. Prueba los componentes combinados para ejecutar un CU.

    2. Para verificar y descubrir errores en las especificaciones de las interfaces de las clases.

- **De sistema**

    1. Prueba el software funcionando como un todo.

    2. Aceptable para cuando el software se encuentra en la Fase de Construcción.

    3. Trata de probar que los objetivos para los que fue construida la aplicación no se cumplen en su totalidad y que por tanto hay que cambiar cosas en la aplicación.

    4. Se usa como base los objetivos originales.

    5. No existe un método en específico, sino que se dan lineamientos, a la hora de preparar los casos de prueba.

6. Se finaliza cuando se cumplieron los meses o las semanas programadas y se han hallado N errores.

- **De liberación**

1. Prueba el software funcionando como un todo.
2. Trata de probar que los objetivos para los que fue construida la aplicación no se cumplen en su totalidad y que por tanto el software no está en condiciones de ser presentado al cliente y hay que hacer modificaciones a la aplicación.
3. Se realiza cuando el software funciona como un todo y está prácticamente listo para ser presentado al cliente.

- **De aceptación**

1. Prueba el software funcionando como un todo, se realiza dándole un uso real a la aplicación y es realizada por parte de los clientes, los errores que se encuentren en la misma son reportados como defectos.

- **De regresión**

1. Pruebas orientadas a descubrir la causa de nuevos errores.

Las pruebas de sistema están divididas en dos grandes grupos:

- **Funcionales:** Enfocadas a los requisitos funcionales del software, a su interacción con el cliente de la forma que ha sido pactada.
- **No funcionales:** Enfocadas a los requisitos no funcionales del proyecto. Desglosadas en un número de pruebas definidas como:

1. **Prueba de seguridad:** Los sistemas y políticas de seguridad son analizados exhaustivamente con el fin de encontrar fallos de seguridad, tanto en el diseño, como en la implementación de la aplicación.

2. **Prueba de disponibilidad y red:** Verifica de manera continua que el entorno está funcionando como espera y su disponibilidad está dentro de los límites que han sido previamente establecidos.

3. **Prueba de rendimiento o carga:** Se ejecutan tanto para determinar como responde un sistema ante una cierta carga, como para validar otros atributos relacionados con la calidad, como pueden ser la escalabilidad o el uso de recursos entre otros.

4. **Prueba de compatibilidad:** Ayudan a determinar si el producto funcionará correctamente con otro hardware y software en el entorno pretendido.

5. **Prueba de resistencia o estrés:** Consiste en poner n usuarios virtuales accediendo concurrentemente durante varias horas y verificar que la aplicación responde a las peticiones en un tiempo menor a t.

6. **Prueba de usabilidad:** Asegurar que se identifiquen y corrijan a tiempo los fallos en la interfaz gracias al seguimiento de los comentarios de los usuarios finales.

7. **Prueba de fiabilidad:** Verifica la probabilidad de que un programa realice su objetivo satisfactoriamente (sin fallos) en un determinado periodo de tiempo y en un entorno concreto (denominado perfil operacional).

1.1.3 Framework analizados para el desarrollo de pruebas unitarias

Zend Framework (6)

Es un framework de código abierto, orientado a objeto, donde cada componente tiene una baja dependencia con respecto a otros componentes lo cual permite a los desarrolladores utilizar los componentes por separado. Entre estas herramientas podemos encontrar:

- Componentes modelo-vista-controlador (MVC)
- Abstracción a Base de Datos
- WebServices
- AJAX

Entre las facilidades que aporta el uso de este framework encontramos:

- Simplicidad en el uso
- Códigos más estables y con menos probabilidad de error
- Simplicidad en el mantenimiento del código

Este framework insiste fundamentalmente en la calidad del código, a través de pruebas unitarias, utilizando PHPUnit. (7)

Symfony (8)

Es un framework para construir aplicaciones web con PHP. Su licencia es de tipo software libre. Contiene un conjunto de herramientas y utilidades que simplifican el desarrollo de las aplicaciones web.

Symfony emplea el tradicional patrón de diseño MVC para separar las distintas partes de una aplicación web. La vista convierte la información obtenida por el modelo en páginas web a las que acceden los usuarios. El controlador es el encargado de ordenar los demás elementos y convertir las peticiones del usuario en operaciones sobre el modelo y la vista.

Entre sus características principales se encuentran:

- Facilita la internacionalización
- Uso de plantillas en la capa de presentación
- Incorpora Lime, un sistema propio de Symfony para realizar pruebas unitarias
- Validación automatizada y relleno automático de datos en los formularios
- Usa técnicas de escape para evitar inyección de código
- Sistema de caché eficiente
- Paginación automatizada
- Los plugins, las factorías (patrón de diseño *"Factory"*) y los *"mixin"* permiten realizar extensiones a medida de Symfony
- Las interacciones con Ajax son muy fáciles de implementar mediante los *helpers* que permiten encapsular los efectos JavaScript compatibles con todos los navegadores en una única línea de código

1.3. Metodologías ágiles

En la década de los noventa surgieron un grupo de metodologías ágiles con características diferentes a las llamadas tradicionales. El término ágil surge en Utah, Estados Unidos en febrero

del 2001 como resultado de en una reunión realizada con las participación de 17 especialistas y expertos de la industria del software, incluyendo a varios de los creadores o impulsores de las metodologías de software. Producto de esta reunión surge el Manifiesto Ágil un documento que tiene explicito un resumen de las concepciones de los procedimientos ágiles. (9)

El término ágil surge a partir del hecho de que estas prácticas cuentan con la habilidad de responder de forma versátil al cambio para lograr mayores beneficios. Sus prácticas y reglas son muy simples y constituyen un nuevo enfoque en el desarrollo de software.

Principios del Manifiesto Ágil (10)

- Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.
- Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.
- Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los períodos breves.
- Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.
- Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.
- La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.
- El software que funciona es la principal medida del progreso.

- Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica enaltece la agilidad.
- La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos que se auto-organizan.
- En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia.

Entre las metodologías ágiles más conocidas y destacadas encontramos:

- Programación eXtrema (XP)
- Scrum
- Crystal Clear y Crystal Methods
- Evolutionary Project Management (Evo)
- Feature Driven Development (FDD)
- Adaptive Software Development (ASD)
- Lean Development (LD) y Lean Software Development (LSD)
- Microsoft Solution Framework (MSF)

1.3.1. Metodología ágil Scrum

Scrum es un proceso ágil y liviano que sirve para administrar y controlar el desarrollo de software.

El desarrollo se realiza en forma iterativa e incremental. Producto de cada iteración se genera una pieza de software ejecutable que incorpora nuevas funcionalidades. Se utiliza en conjunto con otras prácticas de ingeniería de software como Proceso Unificado de Desarrollo de Software (RUP) o XP.

El principal objetivo de esta metodología de gestión de proyectos es elevar al máximo la productividad de un equipo y priorizar el trabajo en función del valor que tenga para el negocio, maximizando la utilidad de lo que se construye y el retorno de inversión. Puede adaptarse a los cambios en los requerimientos. Los requerimientos y las prioridades se revisan y ajustan durante el proyecto en intervalos muy cortos y regulares. El equipo de desarrollo se centra en construir software de calidad.

Características de Scrum

- Tres roles básicos: *Product Owner* (propietario del producto), ScrumMaster (maestro scrum), y *Project Team* (el equipo)
- *Product Backlog* (listado de tareas a desarrollar)
- Iteraciones de un período de tiempo definido (Sprint), entregar incrementos de funcionalidad potencialmente distribuidles al final de cada iteración
- Sprint Backlog (listado de tareas para el Sprint)
- Reunión de planificación del Sprint y reunión de Scrum diaria
- Revisión de cada Sprint
- Retrospectivas

Ventajas

Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar. Además de esto pueden señalarse las siguientes ventajas:

- Entrega de un producto funcional al finalizar cada Sprint
- Posibilidad de ajustar la funcionalidad en base a la necesidad de negocio del cliente
- Visualización del proyecto día a día
- Alcance acotado y viable
- Equipos integrados y comprometidos con el proyecto, toda vez que ellos definieron el alcance y se auto-administran

1.3.2. Metodología ágil XP

Su filosofía es satisfacer las necesidades del cliente, por eso lo integra como parte del equipo de desarrollo. Está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, dónde la comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes.

Inicialmente creada para el desarrollo de aplicaciones dónde el cliente no sabe muy bien lo que quiere, lo que provoca un cambio constante en los requisitos que debe cumplir la aplicación .Los cambios de requisitos sobre la marcha son un aspecto natural e inevitable en el desarrollo de proyectos.

Características de XP

- Desarrollo iterativo e incremental: pequeñas mejoras.

- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión.
- Programación en parejas: el código es revisado y discutido mientras se escribe.
- Frecuente integración del equipo de programación con el cliente o usuario. El cliente trabaja junto al equipo de desarrollo.
- Corrección de todos los errores: hacer entregas frecuentes para revisar lo que se esta produciendo antes de añadir nueva funcionalidad al sistema.
- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar así su legibilidad sin modificar su comportamiento.
- Propiedad del código compartida: este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores sean detectados.
- Simplicidad en el código: es la mejor manera de que las cosas funcionen. Cuando todo funcione se podrá añadir funcionalidad si es necesario. La programación extrema apuesta que es más sencillo hacer algo simple y tener un poco de trabajo extra para cambiarlo si se requiere, que realizar algo complicado y quizás nunca utilizarlo.

Ventajas (11)

- Programación organizada
- Menor taza de errores
- Satisfacción del programador

Beneficios (11)

- El cliente tiene el control sobre las prioridades
- Se hacen pruebas continuas durante el proyecto
- XP es mejor utilizarla en la implementación de nuevas tecnologías donde los requerimientos cambian rápidamente

1.3.3. MA-GMPR-UR2.

La metodología MA-GMPR-UR2 surge en la Universidad de las Ciencias Informáticas como parte de los trabajos de diploma. En el año 2007 la estudiante Malay Rodríguez Villar realizó una propuesta llamada MA-MRV-R1. Dicha propuesta pretendía darle solución a los problemas en el proceso productivo de la Facultad 7 introduciendo el entorno ágil al proceso productivo.

Las metodologías ágiles analizadas en la propuesta son las metodologías: SCRUM, para la planificación de los proyectos y XP para llevar a cabo el proceso de desarrollo del proyecto seleccionando las mejores prácticas.

En junio de 2008 como parte de las recomendaciones de esta tesis de continuar la investigación, se realiza la propuesta MA-GMUR-UR2 donde ya se plantea el proceso completo de fusionar las metodologías SCRUM y XP.

Esta nueva propuesta está siendo aplicada por proyectos productivos de la facultad 10 y en los proyectos que corresponde a la Dirección de Informatización.

Esta metodología surge por la necesidad de mejorar el proceso de desarrollo, que se veía obstruido por un expediente de proyecto hecho solo con RUP y mucha documentación para llenar. Era necesaria una buena documentación pero a la vez dedicar mucho tiempo para el trabajo de los desarrolladores. El exceso de documentación que proponía RUP simplemente

agotaba a los equipos de personas y hacia que los resultados obtenidos por estos fuesen realmente escasos.

MA-GMPR-UR2 (Scrum + XP) metodología ágil propuesta por la facultad 10, cuenta en su totalidad con 4 fases.

- Planificación – Definición
- Desarrollo
- Entrega
- Mantenimiento

La figura 1 representa las fases desarrollada por la metodología MA-GMPR-UR2.

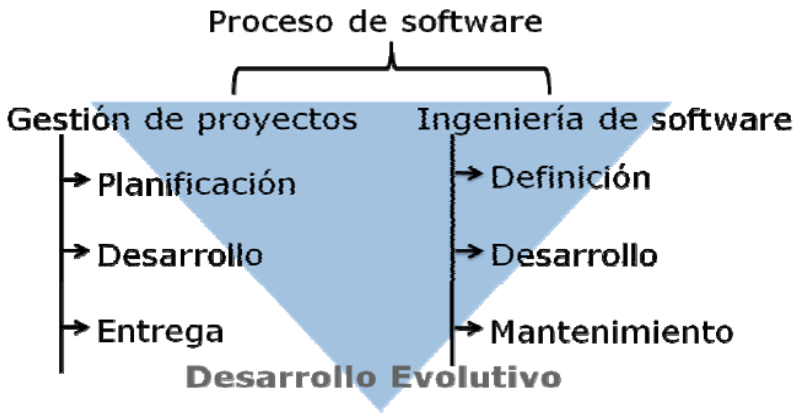

Figura 1. Fases de la metodología MA-GMPR-UR2 (12)

Planificación

En la planificación se definen los objetivos así como los medios para alcanzarlos. Esta fase reduce los riesgos y disminuye al mínimo los problemas potenciales y prepara al grupo de trabajo para hacer frente a las eventualidades que se pueden presentar.

Durante la primera parte de la fase de planificación se realiza la entrevista con el cliente a fin de saber lo que este requiere. Esta primera parte es de vital importancia puesto que desde el primer inicio se le va informando como va a ser su interacción con el desarrollo del futuro sistema.

La definición del Sprint (iteración) es un elemento fundamental y que se genera a partir de la lista de reserva del producto (LRP). Esta no es más que una lista priorizada del trabajo que se realizará en el proyecto la cual esta sujeta a cambios mientras se va conociendo lo que se realiza en el proyecto y a medida que el cliente esta consiente de lo que quiere.

Un Sprint no es más que un ciclo iterativo donde se desarrolla o mejora una funcionalidad del software que se está produciendo. En ella se manejan tanto requerimientos como otras variables como tiempo, recursos, conocimiento y tecnología.

Historias de usuarios (HU) (13)

Una Historia de Usuario es una representación de un requerimiento de software escrito en una o dos frases utilizando el lenguaje común del usuario. Estas con escritas por los clientes y son usadas para la especificación de requerimientos. Las HU son la guía para las pruebas de aceptación, las cuales constituyen el elemento clave en XP.

Las HU permiten dividir los proyectos en pequeños entregables y son buenas para proyectos con requerimientos volátiles o no muy claros puesto que permiten responder rápidamente a los requerimientos cambiantes.

Las HU al ser muy cortas representan requisitos del modelo de negocio que se implementan rápidamente y necesitan poco mantenimiento.

En esta primera fase se definen cuales son las HU que se van a implementar. Como parte del juego de la planificación se ordenan las HU según prioridad y esfuerzo, y se define el contenido de la entrega, definiendo así que HU se implementará primero y cual después según su importancia para el cliente.

Desarrollo

El propósito de la fase de desarrollo es implementar un sistema listo para la entrega en una serie de iteraciones de 60 días. Como parte de las actividades se realiza la planificación de la iteración, reuniones de coordinación, revisión de la iteración y la realización de las pruebas de aceptación del sistema.

Como conclusión de cada semana se realiza una reunión donde se irá comprobando por cada Historia de Usuario que se ha hecho, que falta por realizar y los problemas que se estén presentando para el desarrollo de la implementación.

El cronograma de producción ayuda a recoger las actividades que se van a ser realizadas en el equipo de desarrollo durante la iteración y por cada iteración se planifica un cronograma.

Entrega

En esta fase se realiza la entrega del software y la documentación del sistema implementado.

Como parte clímax de esta fase se encuentra el entrenamiento o capacitación del cliente en el entorno de la aplicación. Un elemento importante para esta actividad es la Plantilla Manual de Usuario la cual contiene una guía de los pasos necesarios y de conceptos importantes de consulta en caso de dudas de los usuarios a la hora de manipular el sistema.

Mantenimiento

En esta fase se realizan las actividades relacionadas con el soporte del software y se generan los documentos relacionados con los cambios que puedan ocurrir en el mismo.

Ventajas de aplicar MA-GMPR-UR2

- Crea un ambiente diverso de Ingeniería de Software (IGSW), donde conviven más de una metodología.
- Promueve más desarrollo en menos tiempo, y documentación de calidad.
- Los entornos donde se use la metodología serán menos complejos, con menos roles y más objetividad.

1.4. Las metodologías ágiles y las pruebas

El desarrollo temprano y sistemático de casos de prueba es importante cuando se aplican metodologías ágiles. Esto se debe a que en este tipo de metodologías pueden producirse transformaciones en los requerimientos en tiempos muy cortos.

Durante el desarrollo del proceso de pruebas uno de los principales objetivos que se persiguen es evitar disminución de la calidad, producto de iteraciones tan cortas.

La fase de prueba aplicada a metodologías ágiles tiene peculiaridades, partiendo como base de que el proceso de pruebas es decididamente un proceso no-ágil, la tarea primordial del encargado de las pruebas debe estar encaminada a minimizar lo más posible el tiempo en que se desarrollen las pruebas.

Un procedimiento de prueba contiene los pasos previstos que llevan a la construcción exitosa del software. Detalla ¿que hacer? para efectuar la prueba, ¿cuándo se deben planificar? ¿y cuánto esfuerzo, tiempo y recursos se van a requerir?

1.5. Conclusiones

Como parte de este capítulo se logró realizar el estudio del arte de la calidad y de los distintos tipos de pruebas de calidad que se realizan en el mundo del software actual. Se realizó además una breve descripción de las metodologías ágiles Scrum y XP, así como una descripción de la metodología ágil MA-GMPR-UR2.

CAPÍTULO II. PROPUESTA DE PROCEDIMIENTO PARA LAS PRUEBAS DE CALIDAD

2.1. Introducción

Durante el desarrollo del siguiente capítulo, se exponen la propuesta de solución de la investigación, elaborándose un procedimiento de pruebas de calidad para la metodología MA-GMUR-UR2.Se realiza un análisis crítico de las pruebas desarrolladas en dicha metodología y su aplicación en los proyectos ServiceDesk (Sistema de Gestión de Reportes) y dotproject SXP (módulo encargado de analizar los reportes en formato XML que llegan al servidor) desarrollados en la facultad 10.

2.2. Descripción del proceso de Pruebas de la metodología MA-GMUR-UR2

Para armar un rompecabezas se buscan primero las similitudes entre las figuras y las ranuras de los pedazos de cartón y posteriormente se unen en un todo. Es imposible armar figuras que no tengan ningún sentido pues al final se producirá un rompecabezas sin lógica. Producto de cada iteración se obtienen pequeñas entregas de programas ejecutables que cuando se integren proveerán un resultado para el cliente. La idea es producir de manera rápida versiones del sistema que sean operativas.

Las entregas deben ser lo más pequeñas o cortas posibles, estas deben contener los requisitos más valiosos del sistema. Como requisito fundamental las entregas están compuestas por Historias de Usuario.

El *tester* es el encargado de ayudar al cliente a escribir las pruebas funcionales. Ejecuta las pruebas regularmente, difunde los resultados en el equipo y es responsable de las herramientas de soporte para pruebas.

En la metodología la producción de código esta dirigida por las pruebas unitarias. Las pruebas unitarias se escriben antes que el código y son ejecutadas constantemente antes ante cualquier

modificación que se realice al sistema. Las pruebas funcionales o pruebas de aceptación son realizadas por el cliente y mediante ellas se valida que la aplicación hace lo que el especificó.

Las pruebas unitarias las realizan los programadores y las pruebas funcionales el cliente.

En la planilla Casos de Prueba de Aceptación el desarrollador escribe las pruebas realizadas según la HU a comprobar y validar. Esta es la manera en que se puede saber si el producto esta listo para ser o no liberado.

2.2.1. Principales inconvenientes del proceso de pruebas de la metodología MA-GMUR-UR2

- **Pruebas unitarias manuales**

En los proyectos donde se aplica la metodología actualmente las pruebas unitarias se realizan de manera manual lo cual es un trabajo muy tedioso y que a la larga dificulta el desarrollo posterior de estas pruebas. Esto se debe principalmente a la falta de conocimiento respecto al tema en cuestión. Con cambios constantes en los requisitos, las pruebas deben poder repetirse, independientes y auto-chequearse lo cual no se puede hacer si se realizan de manera manual.

Auto-chequeables: las pruebas deben saber todas las salidas del programa, saber si están pasando.

Repetitivas: las pruebas deben ser ejecutadas varias veces sin ningún cambio en sus resultados.

Independientes: estas deben ser independientes de otras pruebas para que ellas se ejecuten solas.

- **Pruebas se aceptación insuficientes**

Las pruebas de aceptación propuestas por la metodología son insuficientes puesto que solo recoge la opinión del usuario que interactúa con el grupo de desarrollo. Esto unido a la falta de conocimiento de la metodología impide que se tomen en cuenta la opinión de los usuarios finales de la aplicación. Es difícil saber como piensan muchas personas si antes no te sientas y compartes con ellos. Además muchas veces son los mismos programadores los que toman el rol del cliente y realizan sus propias pruebas de aceptación sin consultar al cliente.

Al realizar las entrevistas a los gerentes y clientes de los proyectos ServiceDesk y dotproject SXP se detectaron algunas insuficiencias del proceso de prueba de calidad con la utilización de la metodología MA-GMUR-UR2. Para consultar la entrevista ver el Anexo A.5.

- **Conocimiento escaso de la metodología**

La Universidad de las Ciencias Informáticas imparte como parte de las asignaturas perfil de la carrera la asignatura “Ingeniería de Software” y como metodología a estudiar RUP. Esto implica que los estudiantes que trabajan en los proyectos la base que tienen de utilización de metodologías ligeras son nulos. Aun el hecho de impartir un curso de metodologías ágiles no garantiza que se aplique tal y como se plantea pues los efectos RUP son muy influyentes. Simplemente se trabaja muchas veces sin tener en cuenta al cliente o haciendo el papel del cliente. Otro de los efectos del desconocimiento de la metodología es la realización tardía de los casos de prueba de aceptación. Estas no se deben hacer ya cuando el software este casi listo o terminado porque entonces de que valen las iteraciones.

- **Desconocimiento de la etapa de prueba por parte de los programadores y poco control de los roles**

El efecto de esto es devastador. Muchos simplemente quieren producir. Pero lo más importante que plantean las metodologías ágiles es la inclusión del cliente en lo que se va desarrollando. En los procesos ágiles cada cual tiene una función. El cliente realiza las pruebas de aceptación no el programador, ni aun el *encargado de las pruebas (Tester)*.

- **Documentación escasa sobre las deficiencias que va presentando el software por iteraciones por parte del cliente**

Lo anteriormente expuesto conlleva a que haya una mala documentación de los errores que se van produciendo por iteraciones, lo cual evidencia que el cliente casi no tiene conocimiento de lo que se va realizando en el proyecto. Esto a la larga puede traer graves consecuencias como el hecho de que al final el cliente no se sienta familiarizado con la aplicación propuesta y simplemente rechace el contrato.

2.3. Descripción de la propuesta de solución

De acuerdo al estudio realizado del proceso de prueba que propone la metodología MA-GMUR-UR2 y su aplicación en los proyectos ServiceDesk y dotproject SXP, se concluye que el proceso de prueba es insuficiente para garantizar la calidad de los productos de software debido a que solo proponen las pruebas unitarias y las pruebas de aceptación, involucrando solo a los clientes que trabajan con el equipo de desarrollo, no teniendo en cuenta la opinión de los usuarios finales de la aplicación. Estas pruebas no garantizan revelar las causas de nuevos errores, insuficiencias de funcionalidad, o discrepancias funcionales con respecto al comportamiento que se esperaba del software, provocados por cambios recientes realizados en partes de la aplicación que anteriormente no era vulnerable a este tipo de error.

Por tanto se propone un procedimiento de pruebas de calidad para la metodología MA-GMUR-UR2 que garantice la eficiencia del proceso de prueba y la calidad del producto final.

2.4. Procedimiento de pruebas de calidad para la metodología MA-GMUR-UR2

La propuesta de solución se centra en dos aspectos fundamentales:

1. Plan de pruebas
2. Diseño de casos de prueba

- **Plan de Prueba**

Los Planes de Pruebas están formados por un conjunto de pruebas. Estos describen en qué consisten las pruebas, como se realizan y cual es el resultado de las mismas.

La propuesta de solución define la realización de las pruebas unitarias, de aceptación, integración y regresión para los entornos de trabajo que utilicen la metodología MA-GMUR-UR2.

2.4.1. Pruebas unitarias

Son pruebas orientadas a descubrir errores en el código del programa que se realiza. Estas pruebas se realizan a unidades (módulos) del sistema. Este tipo de prueba verifica si una unidad funciona correctamente sin tener en cuenta su relación con las otras partes del sistema.

Estas pruebas no descubrirán todos los errores del código, estas solo prueban que el segmento de código que se revisa está lógicamente bien. No descubrirán errores de integración.

Encargados de la prueba: Programadores.

Momento en que se ejecuta la prueba: Una vez terminada la implementación de las HU planificadas para la iteración se realizan las pruebas unitarias escritas previamente.

Como resultado de estas pruebas se generan las Plantillas de Caso de Prueba de Unidad.

Estas pruebas tienen la ventajas que pueden ser realizadas de manera automatizadas utilizando framework como Symfony y Zend Framework.

2.4.2. Pruebas de integración

Fusionar todos los módulos y el programa en su conjunto puede tener un efecto catastrófico en el uso de las metodologías ágiles pues podría provocar un conjunto de fallos y así como la dificultad para identificar el módulo o módulos donde los errores se producen. Aun cuando los

módulos de un programa funcionen bien por separado es necesario probarlos conjuntamente ya que un módulo puede tener un efecto adverso o inadvertido sobre otro módulo.

El objetivo de la prueba de integración no es más que tomar el resultado de la prueba de unidad y construir una estructura de programa que este de acuerdo con lo que se quiere en el diseño. El foco de atención en este tipo de pruebas es el diseño y la construcción de la arquitectura del software.

Consiste en realizar pruebas que verifiquen que un gran conjunto de partes del software funcionan. Las pruebas de integración son llamadas algunas veces pruebas de integración o testeo.

Existen dos tipos de Integración:

- Integración incremental: El programa se construye y se prueba en pequeños segmentos.
- Integración no incremental: Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto.

La prueba de Integración incremental se planifica para cada una de las iteraciones del proyecto y se realizan a cada uno de los pequeños entregables que se obtienen. Más adelante cuando el sistema está completo se realizan las pruebas de Integración no incremental.

Encargados de la prueba: Programadores.

Momento en que se ejecuta la prueba: Terminadas las pruebas unitarias de procederá a la realización de las pruebas de integración donde se probarán todos las HU que se implementaron durante la iteración. Durante la iteración siguiente se probarán en conjunto los resultados de la iteración-1 y la iteración en cuestión.

Como resultado de la etapa de prueba se generan las Plantillas de Caso de Prueba de Integración incremental y no incremental.

2.4.3. Pruebas de aceptación

Las pruebas de aceptación son realizadas por el cliente y son básicamente pruebas funcionales, sobre el sistema completo. Estas pruebas se realizan sobre el producto terminado e integrado o sobre una versión del producto o iteración funcional. El objetivo de estas pruebas es validar que el sistema cumple con el funcionamiento esperado y permitir al usuario determinar su aceptación, desde el punto de vista de su funcionalidad y rendimiento. La validación se consigue mediante la realización de pruebas de caja negra.

Actualmente la metodología solo realiza las pruebas de aceptación alfa, por lo que se propone las pruebas de aceptación beta que permiten controlar el comportamiento del sistema y la satisfacción de los usuarios ante el mismo. (14)

- Técnica prueba de aceptación Alfa: Se lleva a cabo por el cliente que interactúa con el equipo de desarrollo. Se utiliza el software de forma natural con el desarrollador como observador del usuario. Las pruebas alfa se llevan a cabo en un entorno controlado donde el cliente va maniobrando el producto y el desarrollador va documentando los resultados. Como resultado de estas pruebas se obtiene la Plantilla de Caso de Prueba de Aceptación Alfa donde intervienen los roles de *tester*, desarrollador y cliente.

- Técnica prueba aceptación Beta: Se lleva a cabo por los usuarios finales del software en los entornos de trabajos de los clientes. En esta prueba el desarrollador no está presente por lo que es el cliente quien registra todos los problemas reales que encuentra durante la prueba y los informa al desarrollador. Como resultado de esta prueba se genera la Plantilla de Caso de Prueba de Aceptación Beta.

Las pruebas de aceptación se realizan por Historia de Usuario. Una HU puede tener más de un caso de prueba, según su complejidad. La plantilla de la estructura de una HU se representa en el Anexo A.1.

Encargados de la prueba de aceptación alfa: Cliente.

Momento en que se ejecuta la prueba: Una vez concluida la integración se realizan las pruebas de aceptación alfa donde el cliente que interactúa con la aplicación prueba el software y ve si este realiza lo que inicialmente se especificó. Como parte de este proceso se le entrega una versión de la aplicación y las plantillas de casos de pruebas de aceptación que debe llenar.

Como resultado de la etapa de prueba se genera la Plantilla de Caso de Prueba de Aceptación Alfa.

Encargados de la prueba de aceptación beta: Usuarios finales.

Momento en que se ejecuta la prueba: Estas pruebas se realizan seguidamente. Se les da a los usuarios finales de la aplicación un entregable listo y se van tomando las impresiones de esto con respecto a la aplicación. Como resultado de esto se recogen una serie de datos útiles para la posterior implementación del sistema.

Como resultado de la etapa de prueba se generan las Plantillas de Caso de Prueba de Aceptación Beta.

2.4.4. Pruebas de regresión

Las pruebas de regresión son pruebas de software orientadas a revelar las causas de nuevos errores, insuficiencias de funcionalidad, o discrepancias funcionales con respecto al comportamiento que se esperaba del software, provocados por cambios recientes realizados en partes de la aplicación que anteriormente no era vulnerable a este tipo de error. Casi siempre son errores inesperados que se producen al ocurrir un cambio en el programa.

Estos cambios pueden ser producto de prácticas no adecuadas de control de versiones, como consecuencia del rediseño de la aplicación, falta de atención acerca del ámbito o contexto de producción final y extensibilidad del error que fue corregido. El objetivo fundamental de estas

pruebas es asegurar que los defectos identificados en la ejecución anterior de la prueba se han corregido y que los cambios realizados no han introducido nuevos errores.

En este tipo de metodologías donde los requisitos pueden estar sujetos a cambios este tipo de pruebas es muy importante y deben realizarse de manera constante. Existen varios tipos de regresión:

- **Clasificación de ámbito**

Local: los cambios introducen nuevos errores.

Desenmascarada: los cambios revelan errores previos.

Remota: los cambios vinculan alguna otra parte del programa (módulo) e introducen errores en ella.

- **Clasificación temporal**

Nueva característica: los cambios realizados con respecto a nuevas funcionalidades en la versión introducen errores en otras novedades en la misma versión del software.

Característica preexistente: los cambios realizados con respecto a nuevas funcionalidades introducen errores en funcionalidad existente de previas versiones.

Encargados de la prueba de regresión: Tester.

Momento en que se ejecuta la prueba: Estas pruebas son realizadas cada vez que se produzca un cambio en la aplicación. Cada vez que se realiza una nueva iteración, implementándose nuevas HU se realizan las pruebas de regresión. Una vez concluido el sistema o durante la fase de mantenimiento se realizan esta pruebas con el fin de revelar la causa de nuevos errores ante los cuales la aplicación anteriormente no era vulnerable.

Como resultado de la etapa de prueba se genera la Plantilla de Caso de Prueba de Regresión.

2.5. Descripción del Diseño de Caso de Pruebas

A continuación se detalla la información correspondiente al diseño de casos de pruebas propuesto y se describen las planillas de los casos de prueba.

2.5.1. Descripción de la Plantilla Caso de Prueba Unitaria

La planilla tiene en su encabezado el nombre del tipo de prueba que se diseña, entre los primeros datos que recoge encontramos el nombre de la HU a la que se le realizan las pruebas, el código del caso de prueba, la iteración en la que se realiza la prueba y el nombre del encargado de la prueba. Como parte de la realización del caso de prueba se recoge posteriormente el ambiente con la información sobre la configuración de hardware o software el cuál ejecutará el caso de prueba seguido de una breve descripción de la prueba que se esta realizando y de las condiciones para la ejecución de la misma.

A la hora de documentar la prueba que se realiza se toma en cuenta el nombre de la funcionalidad a probar, el número de veces que se ha chequeado esta prueba (en caso de realizarse por primera vez se pone valor cero), el tipo de prueba según el framework a utilizar, el valor que recibe seguido del valor que se espera que devuelva la prueba y finalmente el valor esperado.

Como parte de las conclusiones del caso de prueba se realizará la evaluación de la prueba que no es más que la descripción por parte del programador de los resultados obtenidos tras haber completado todos los pasos de la prueba.

En una misma Plantilla de Caso de Prueba Unitaria se registran todas las pruebas a las funcionalidades que se le hayan asignado al programador según las HU.

La plantilla de Caso de Prueba Unitaria se representa en el Anexo A.2.

2.5.2. Descripción de la Plantilla Caso de Prueba de Integración no incremental o Caso de Prueba de integración incremental

La planilla tiene en su encabezado el nombre del tipo de prueba de integración que se diseña, si es del tipo incremental o no incremental. Entre los primeros datos que recoge encontramos código del caso de prueba, el nombre del encargado de la prueba, las herramientas o facilidades utilizadas para realizar la prueba, la iteración y el conjunto de HU a probar durante la prueba. Seguidamente realiza una breve descripción de la prueba, se recogen los valores de entrada de la prueba, se explican las condiciones de ejecución necesarias para la ejecución de la prueba y una serie de pasos necesarios para lograr la realización del caso de prueba.

Como parte de las conclusiones del caso de prueba se registrará el resultado de la prueba, un resumen de los defectos encontrados enumerados y se realizará la evaluación de la prueba que no es más que una descripción de lo que el encargado de la prueba debería ver tras haber completado todos los pasos de la prueba.

Para las pruebas de integración no incremental y las pruebas de integración incremental se aplicará la misma plantilla de caso de prueba. La plantilla de Caso de Prueba de Integración no incremental o Caso de Prueba de integración incremental se representa en el Anexo A.2.

2.5.3. Descripción de la Plantilla Caso de Prueba de Aceptación Alfa

La plantilla tiene en su encabezado el nombre del tipo de prueba que se diseña, entre los primeros datos que recoge encontramos código del caso de prueba, nombre de la HU a la que se realiza la prueba, el nombre del encargado de la prueba, una breve descripción de la prueba que se esta realizando, las condiciones de ejecución necesarias para la prueba y una serie de pasos necesarios para lograr la realización de la HU.

Como parte de las conclusiones del caso de prueba se registrará el resultado esperado como parte de la realización del caso de prueba y se realizará la evaluación de la prueba como satisfactoria o no satisfactoria.

La plantilla de Caso de Prueba de Aceptación Alfa se representa en el Anexo A.2.

2.5.4. Descripción de la Plantilla Caso de Prueba de Aceptación Beta

Esta planilla cuenta con dos secciones fundamentales:

- **Primera sección**

En la primera sección de la planilla se encuentra un encabezado con nombre del tipo de prueba que se diseña. Entre los primeros datos que recoge encontramos código del caso de prueba que contiene la información sobre la cantidad de HU a probar, nombre de las HU a la que se realiza la prueba como parte de la entrega de la iteración, el nombre del encargado de la prueba, una breve descripción de la prueba que se esta realizando, las condiciones de ejecución necesarias para la prueba y una serie de pasos necesarios para lograr la realización de la HU.

Como parte de las conclusiones del caso de prueba se registrará el resultado esperado que no es más que un resumen de lo que el cliente espera seguido de una opción para que el cliente que esta realizando la prueba exprese sus comentarios respecto a la prueba. Finalmente se recoge el grado de satisfacción general en un rango del 1 al 5.

- **Segunda sección**

En esta parte de la planilla se recoge la evaluación del cliente con respecto a partes bien especificas del sistema que se esta automatizando. Esta evaluación se realiza mediante un rango del 1 al 5. Estos datos serán tomados en cuenta por los programadores a la hora de implementar y desarrollar la próxima iteración o en caso de ser algo muy relevante para el desarrollo se deberá solucionar de inmediato. Entre estos datos se recogen los siguientes:

En cuanto a las funcionalidades que brinda el sistema: exactitud de la aplicación o precisión así como el buen funcionamiento del mismo.

Se recoge además la opinión del usuario sobre la usabilidad del sistema si este es atractivo, si presenta un fácil manejo o es complejo de manera que dificulte al usuario el aprendizaje del mismo.

Posteriormente se registra la opinión del usuario sobre la fiabilidad del sistema en cuestión con la madurez, atendiendo a si presenta o no información de exigencias de hardware y software.

Como parte final de esta sección se presenta una tabla con la información del rango: 1 para evaluar de muy malo, 2 para evaluar de malo, 3 para evaluar de regular, 4 para evaluar de bueno y 5 para evaluar de excelente.

La plantilla de Caso de Aceptación Beta se representa en el Anexo A.2.

2.5.5. Descripción de la Plantilla Caso de Prueba de Regresión

La plantilla tiene en su encabezado el nombre del tipo de prueba que se diseña. Entre los primeros datos que recoge encontramos código del caso de prueba y el nombre del encargado de la prueba. A continuación se registra la información sobre el ambiente o configuración de hardware y software el cuál ejecutará el caso de prueba. Seguidamente se explica el propósito de la prueba, seguido de una breve descripción de la prueba que se está realizando.

A continuación se documenta el tipo de error que se ha detectado producto de la prueba, la localización del error así como la acción correctiva a utilizar para solucionar el error encontrado.

Como parte de las conclusiones del caso de prueba se registrará el resultado de la prueba y se realizará la evaluación de la prueba que no es más que una descripción de lo que el encargado de la prueba debería ver tras haber completado todos los pasos de la prueba.

En la parte final de la planilla se encuentra la información de los tipos de errores que pueden producirse. En caso de ocurrir un tipo de error que no se encuentre entre los señalados se documentará el tipo de error.

La plantilla de Caso de Prueba de Regresión se representa en el Anexo A.2.

2.6. Documento garantía de la Calidad

Como parte de la propuesta de solución se propone el Documento Garantía de la Calidad que tiene como objetivo proveer de los resultados de las pruebas realizadas al producto, prueba por prueba, iteración por iteración. Al tener documentadas las pruebas realizadas al producto es más fácil encontrar la causa de errores futuros lo cual facilita el trabajo del equipo de soporte y mantenimiento. La estructura de dicha plantilla se representa en el Anexo A.2.

2.7. Automatización de las pruebas

El propósito de la implementación de pruebas es automatizar los procedimientos de prueba, creando componentes de pruebas. Estos usan por lo general grandes cantidades de datos de entrada y generan grandes cantidades de datos de salida.

La automatización de pruebas es uno de los mayores avances en la programación desde la invención de la orientación a objetos. En el desarrollo de aplicaciones web el probar correctamente la aplicación es una ocupación bastante tediosa.

Con requisitos cambiantes es muy probable que aparezcan nuevos errores por lo cual la automatización de las pruebas es recomendable y útil para crear un entorno de desarrollo satisfactorio.

Las pruebas automatizadas garantizan que aun cuando el código interno de la aplicación cambia constantemente los cambios no introducen errores en el funcionamiento de la aplicación. Pueden llegar a reemplazar la documentación técnica de la aplicación, ya que ilustran de forma clara el

funcionamiento de la aplicación. Un buen conjunto de pruebas muestra la salida que produce la aplicación para una serie de entradas de prueba, por lo que es suficiente para entender el propósito de cada método.

En la propuesta de solución se plantea la automatización de las pruebas unitarias utilizando la metodología MA-GMPR-UR2 para los entornos de desarrollo que utilicen los framework Symfony y Zend Framework.

2.7.1. Automatización de las pruebas unitarias

Las pruebas unitarias garantizan que un único componente de sistema produce una salida correcta para una determinada entrada. Este tipo de pruebas permite validar la forma en la que las funciones y métodos trabajan en un caso particular. Las pruebas unitarias se encargan de un único caso cada vez por lo que un único método puede necesitar varias pruebas unitarias si su actividad varía en con respecto al contexto.

En el ámbito de PHP existen muchos framework para crear pruebas unitarias. Entre los más conocidos podemos encontrar PHPUnit y Symfony.

2.7.2. Automatización de las pruebas unitarias utilizando Symfony (8)

Symfony introduce herramientas y utilidades para automatizar las pruebas las cuales aseguran la calidad de la aplicación incluso cuando el desarrollo de nuevas versiones es muy activo. Incluye su propio framework llamado Lime.

Lime provee soporte para las pruebas unitarias y es más eficiente que otros framework de pruebas de PHP.

Las pruebas unitarias de Symfony son archivos PHP normales cuyo nombre termina en Test.php y que se encuentran en el directorio test/unit/ de la aplicación. El objeto lime_test de Symfony

dispone de un gran número de métodos para las pruebas unitarias. Para consultar los métodos ver Anexo A.3. (8)

La figura 2 muestra un conjunto típico de pruebas unitarias para la función Autenticar (). En primer lugar, se instancia el objeto lime_test. Cada prueba unitaria consiste en una llamada a un método de la instancia de lime_test. El último parámetro de estos métodos siempre es una cadena de texto opcional que se utiliza como resultado del método.

```
16  $t = new lime_test(3, new lime_output_color());
17
18  $ldap = new ldap();
19
20  $t->diag('-------------------Progrando la UnaClase.php--------------');
21  $t->diag('    ');
22  $t->isa_ok($ldap->Autenticar('drhernandez','***********'), 'boolean','Chequear tipo de datos esperados');
23  $t->ok($ldap->Autenticar('drhernandez','tuvivesenmi'),'Probando el metodo Autenticar');
24  $t->is($ldap->Autenticar('drhernandez','tuvivesenmi'), 'false','Chequear tipo de datos esperados');
25  $t->diag('    ');
26  $t->diag('-------------------FIN de la prueba-----------------------');
```

Figura 2. Prueba Unitaria en test/unit/LdapTest.php

Para ejecutar el conjunto de pruebas, se utiliza la tarea test-unit desde la línea de comandos. El resultado de esta tarea en la línea de comandos es muy explícito, lo que permite localizar fácilmente las pruebas que han fallado y las que se han ejecutado correctamente. La figura 3 muestra el resultado del ejemplo anterior.

```
Z:\>D:\wamp\bin\php\php5.2.5\php.exe symfony test-unit Ldap
1..3
# -------------------Progrando la UnaClase.php--------------
#
ok 1- Chequear tipo de datos esperados
ok 2- Probando el metodo Autenticar
not ok 3- Chequear tipo de datos esperados
#  Failed test <.lib\symfony\verdor\lime\lime.php at line 103>
#         got: false
#    expected: 'false'
#
#  -------------------FIN de la prueba-----------------------
 Looks like you failed 1 tests of 3
```

Figura 3. Resultado de la Prueba Unitaria en test/unit/LdapTest.php

2.7.3. Automatización de las pruebas unitarias utilizando Zend Framework (6)

PHPUnit Testing es un procedimiento para validar unidades de código fuente. Este producto provee la forma de hacer las prueba de PHPUnit testing de dos formas, la primera es generando casos de pruebas y suits de prueba y la segunda es la interfaz para probar y analizar los resultados de las pruebas.

Para ver ejemplos de sentencias o métodos para las Pruebas Unitarias de Zend Framework ver Anexo A.3. A continuación veamos un ejemplo. Se prueba la unidad calculadora (Calculator). En la figura 4 se representa el archivo “Calculator.php” donde podemos ver funciones simples de cálculo.

```
<?php
class Calculator{
  public function add($a, $b) {
      return $a + $a;
  }

  public function multiply ($a, $b) {
      return $a * $b;
  }

  public function divide ($a, $b) {
      if($b == null) {
         trow new Exception ("Division por cero");
      }
      return $a/$b;
  }

  public subtract ($a,$b) {
      return $a - $b;
  }
}


?>
```

Figura 4. Clase Calculator.php

Se puede probar cada una de estas funciones y ver si trabajan correctamente en situaciones diferentes. Para hacer esto se necesita crear un caso de prueba.

Un caso de prueba se puede crear haciendo clic derecho en la carpeta del proyecto o desde el menú principal. Seleccione new/PHPUnit Test Case y se abre una ventana. A continuación se selecciona la carpeta origen y la superclase. Por defecto se puede usar la misma del caso de prueba del framework. Luego se selecciona el elemento a probar, la clase y el nombre del fichero. Se da clic en la opción "*Finish*" apareciendo el área del código del caso de prueba creado con el esqueleto de la prueba. En la figura 5 se representa el área del código del caso de prueba.

```
55  /**[]
58  protected function testAdd()
59  {
60          //TODO Auto-generated CalculatorTest2->testAdd()
61          this->markTestIncomplete("Add test not implemented")
62
63          this->Calculator->Add(/* parameters */)
64  }
65
66
67  /**[]
70  publicfunction testDivide()
71  {
72          //TODO Auto-generated CalculatorTest2->testDivide()
73          this->markTestIncomplete("Add test not implemented")
74
75          this->Calculator->Divide(/* parameters */)
76
77  }
78
79  /**[]
82  publicfunction testMultiply()
83  {
84          //TODO Auto-generated CalculatorTest2->testMultiply()
85          this->markTestIncomplete("Add test not implemented")
86
87          this->Calculator->Multiply(/* parameters */)
88
89  }
90
91  /**[]
```

Figura 5. Área del código de la prueba

Para correr el Caso se Prueba simplemente se hace clic derecho sobre el fichero en la vista del explorador, se selecciona la opción “run as” después la opción PHPUnit test (también se puede hacer por la configuración del “*Run launch*”).

Aplicando el procedimiento al caso de estudio la figura 6 representa como quedarían los casos de prueba.

```
55  /**
56  *Test Calculator->Add()
57  */
58  protected function testAdd()
59  {
60   this->assertEquals( $this->Calculator->add(1,2),3);
61   this->assertEquals( $this->Calculator->add(1,3),5); //4
62   this->assertEquals( $this->Calculator->add(1,3),5); //4
63
64 }
65  /**
66  *Test Calculator->divide()
67  */
68  publicfunction test_divide()
69  {
72          this->assertEquals($this->Calculator->divide(5,1),5);
73          this->assertEquals($this->Calculator->divide(5,0),0); //bad
74          this->assertNull( $this->Calculator->divide(0,5));
75          this->assertSame( $this->Calculator->divide(2,2),this->Calculator->Divide(2,2));
76
77  }
78
79  /**[]
82  publicfunction test_multiply()
83  {
84          this->assertEquals($this->Calculator->multiply(5,0),10);
85          this->assertEquals($this->Calculator->multiply(5,0),0);
86          this->assertSame(0, $this->Calculator->multiply(0,5));
87          this->assertSame(0, $this->Calculator->multiply(5,0));
88          this->assertSame(0, $this->Calculator->multiply(5,1));
89  }
90
91  /**[]
```

Figura 6. Resultado Caso de estudio Clase Calculator

2.8. Conclusiones

Durante el desarrollo del capítulo se elaboró la propuesta de procedimiento para las pruebas de calidad en metodología MA-GMUR-UR2. Se realizó un análisis crítico de las pruebas desarrolladas en la metodología, así como de su aplicación en los proyectos ServiceDesk y dotproject SXP desarrollado en la facultad 10.

CAPÍTULO III. VALIDACIÓN DE LA PROPUESTA DE PROCEDIMIENTO DE PRUEBAS DE CALIDAD PARA LA METODOLOGÍA MA-GMPR-UR2

3.1. Introducción

Para la validación y aceptación de la propuesta de procedimiento para las pruebas de calidad en la metodología MA-GMPR-UR2 presentadas en el Capítulo 2, se utilizó el criterio de un grupo de expertos. Este panel se conformó con especialistas que poseen conocimientos en pruebas de calidad. En el presente capítulo se hará la descripción de los pasos utilizados en la selección del panel de expertos y los resultados obtenidos. Para el proceso de evaluación de la propuesta se tuvo en cuenta el proceso de selección de expertos, elaboración de la encuesta que se aplicó y por último los resultados de la evaluación. La calidad de los resultados depende, sobre todo, del cuidado que se ponga en la elaboración del cuestionario y en la elección de los expertos consultados.

3.2. Proceso de selección de expertos

Se entiende por experto a una persona que tiene conocimiento y experiencia en un campo. Un especialista en una materia. Es capaz de interpretar correctamente las informaciones sobre dicho campo, de ofrecer valoraciones conclusivas de un problema en cuestión y hacer recomendaciones al respecto. En el desarrollo de este proceso se consideraron tres etapas cruciales:

- Determinar la cantidad de expertos.
- Conformar el listado de los expertos.
- Confirmar la participación de los expertos.

Se tiene en cuenta que ningún experto conoce las identidades y respuestas de los otros que componen el grupo, para lograr así que cada uno defienda sus opiniones y en caso de ser erróneas, no habrá pérdida de su prestigio.

3.2.1. Determinar la cantidad de expertos

Se seleccionó una muestra de siete expertos para la confección del panel, teniendo en cuenta el nivel de complejidad del contenido.

3.2.2. Conformar el listado de expertos

La confección del listado de expertos se realizó atendiendo a la posibilidad real de participación de los candidatos, pues todos son profesionales de la UCI que tienen experiencia en la docencia, y en el proceso productivo de la universidad. Poseen además, amplios conocimientos en temas relacionados con el proceso a evaluar, estos son:

- Pruebas de calidad de software.
- Calidad de software.

Existe una serie de cualidades propias de estos especialistas seleccionados, que se tuvieron en cuenta por parte de los autores de esta investigación para la confección del listado. A continuación se relacionan:

- Seriedad.
- Honestidad.
- Sinceridad.
- Responsabilidad.
- Creatividad.

- Capacidad de análisis.

Estas cualidades han permitido que las opiniones brindadas sean confiables y válidas para el objetivo propuesto. (Ver Anexo A.4)

3.2.3. Confirmar participación de expertos

Una vez conformado el listado, se invitó personalmente a cada experto elegido para participar en la evaluación. Allí se les explicó en que consistía el trabajo en general, la propuesta a evaluar y el objetivo de la realización de la encuesta, así como el plazo de entrega. Una vez recibida la respuesta positiva, se estableció el listado final de los expertos, informando a cada especialista su inclusión en el proceso a evaluar y las instrucciones necesarias para contestar las preguntas. De esta forma culmina el proceso de selección, logrando la participación de los siete expertos escogidos.

3.3. Elaboración de la encuesta

Las encuestas se llevan a cabo de una manera anónima. Para la elaboración de la encuesta se tuvieron en cuenta los objetivos que debería cumplir el procedimiento propuesto para su implantación en los proyectos que utilizan la metodología MA-GMPR-UR2. Se les facilitó la posibilidad de modificar aspectos que ellos consideraban necesarios cambiar y presentar su opinión general, a favor o en contra del procedimiento propuesto, con la libertad de expresar todo lo que se pudo obviar en la encuesta. (Ver Anexo A.3). Se les fue a ver a cada uno personalmente, los autores del trabajo de investigación le explicaron detalladamente a cada experto los objetivos y resultados de la propuesta, luego se les dio la encuesta con un plazo de tiempo para entregarla.

La encuesta establece una serie de preguntas que permiten visualizar la posibilidad real de aplicar la propuesta.

3.4. Resultados de la evaluación

El 100% de los expertos considera que el procedimiento propuesto está a la altura de las necesidades de la metodología MA-GMPR-UR2.

Los expertos 1, 2, 6 y 7 argumentan que la propuesta de solución facilita el incremento de las pruebas de calidad en cada una de las iteraciones de la metodología MA-GMPR-UR2 y registra efectivamente la documentación con los resultados de cada una de las pruebas realizadas al producto en cuestión.

El 100% de los expertos consideran que el desarrollo de las pruebas propuestas es lo suficientemente factible a las necesidades de los proyectos productivos que utilicen la metodología MA-GMPR-UR2. El experto 2 considera que el procedimiento se puede adaptar a los proyectos productivos que utilicen la metodología MA-GMPR-UR2, sin afectar en gran medida su tiempo de realización, que es lo que se busca fundamentalmente con la utilización de metodologías ágiles

En la última pregunta aplicada a los expertos, se evalúa una serie de criterios, dándoles una categoría de 1 a 5, donde uno es el mínimo y 5 el máximo. La Tabla 3.2 muestra los resultados de la misma.

Tabla 2. Resultados de las opiniones de los expertos

Criterio/Experto	**Experto 1**	**Experto 2**	**Experto 3**	**Experto 4**	**Experto 5**	**Experto 6**	**Experto 7**
Satisfacción a las necesidades de la metodología MA-GMPR-UR2.	4	5	5	4	5	5	5

Adaptabilidad a los proyectos productivos.	5	4	4	5	4	4	4
Mejora del proceso de prueba.	3	4	5	4	5	5	4
Posibilidad de aplicación.	4	5	4	4	5	3	5

El 71,43% de los expertos evaluó de 5 la satisfacción que tiene esta propuesta a las necesidades de la metodología MA-GMPR-UR2. El resto propuso una calificación de 4.

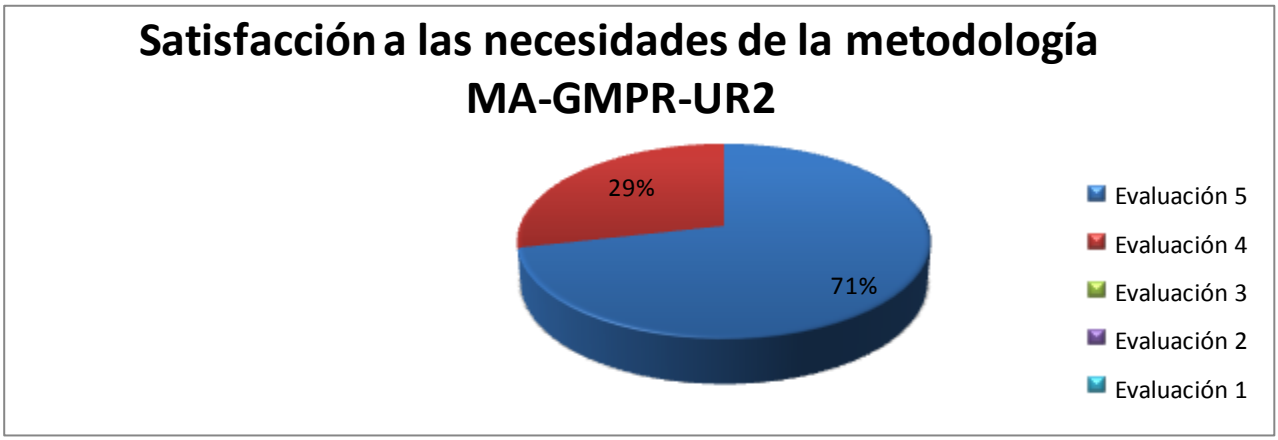

Figura 7. Satisfacción a las necesidades de la metodología MA-GMPR-UR2

El 28,57% de los expertos evaluó de 5 la adaptabilidad de la propuesta a los proyectos productivos. El resto propuso una calificación de 4.

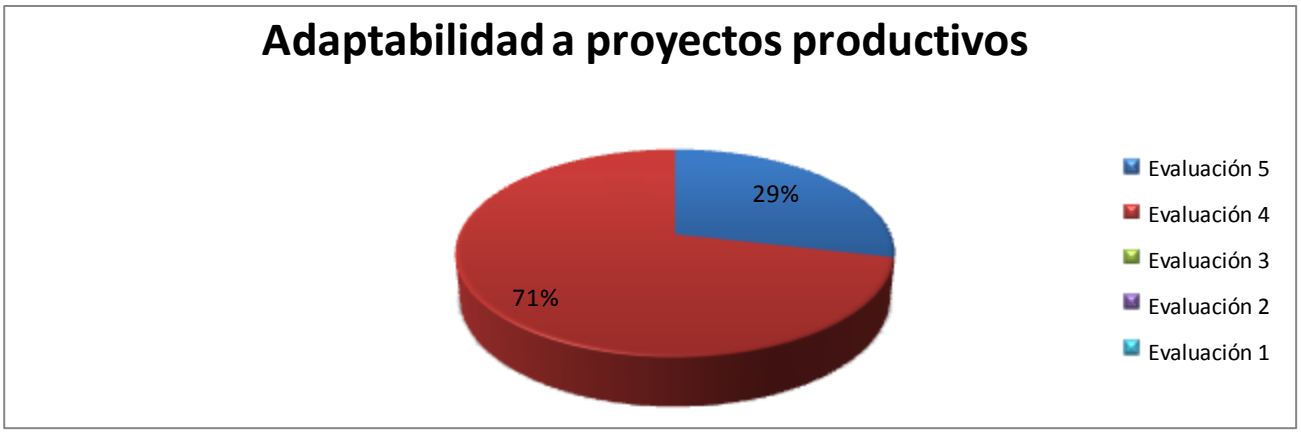

Figura 8. Adaptabilidad a proyectos productivos

El 42,86% de los expertos calificó de 5 la repercusión de la propuesta en la mejora del proceso de prueba, el 42,86% de los expertos calificó de 4 y 14,28 de los expertos calificó de 3.

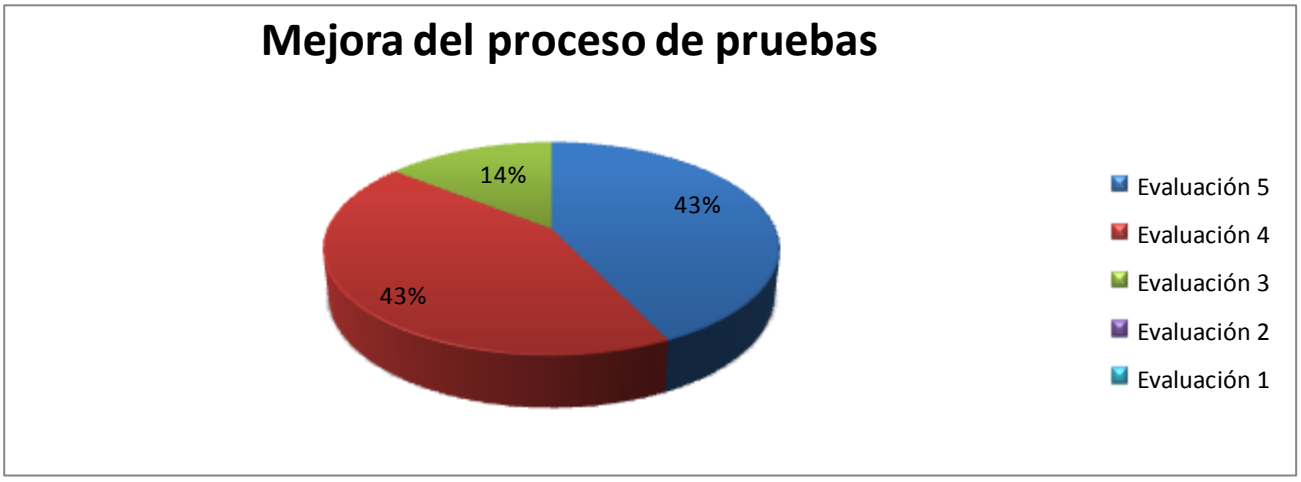

Figura 9. Mejora del proceso de pruebas

El 42,86% de los expertos calificó de 5 la posibilidad de aplicar esta propuesta a los proyectos productivos. Hubo una evaluación de 3 y el resto de 4.

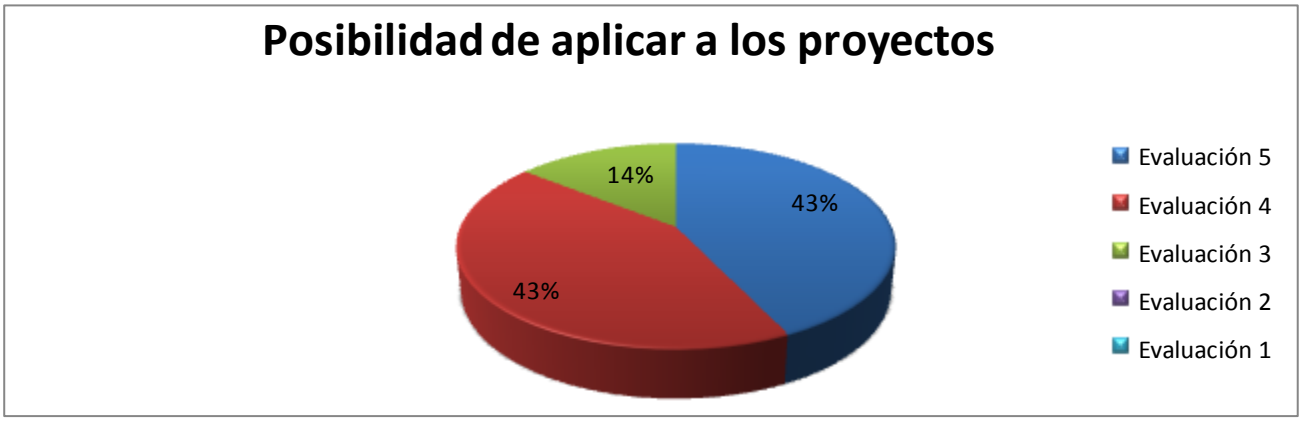

Figura 10. Posibilidad de aplicar a los proyectos

3.5. Conclusiones

Como parte de este capítulo se realizó la validación y aceptación de la propuesta de procedimiento para las pruebas de calidad en la metodología MA-GMPR-UR2 presentadas en el Capítulo 2 mediante la técnica de Panel de Expertos. Se realizó una descripción de los pasos utilizados en la selección del panel de expertos y se realizó una selección los expertos que contaban con pleno conocimiento y experiencia en el área de las pruebas de calidad. Por último se analizaron los datos obtenidos en las entrevistas a los expertos mediante gráficos.

CONCLUSIONES

En este trabajo se demostró que el proceso de pruebas de calidad desarrollado actualmente por la metodología MA-GMPR-UR2 es insuficiente para garantizar la calidad de los productos de software. La propuesta de solución introduce mejoras en el proceso de prueba de calidad para los proyectos que utilicen dicha metodología, que permiten revelar las causas de nuevos errores, insuficiencias de funcionalidad o discrepancias funcionales con respecto al comportamiento que se esperaba del software.

Una vez realizado el análisis de las pruebas de calidad más usadas en la actualidad para el desarrollo de sistemas informáticos y, dentro de ellas las más utilizadas en las metodologías ágiles, específicamente en MA-GMPR-UR2, se concluye que es conveniente incorporar en el procedimiento de pruebas de dicha metodología las pruebas de integración y regresión, por las grandes potencialidades que presentan para garantizar la calidad del producto final y con ello la satisfacción del cliente.

Partiendo de la generación de casos de pruebas, la propuesta de solución que se presenta brinda la posibilidad de documentar las pruebas realizadas al software, siendo más fácil para el equipo de soporte encontrar las causas de errores futuros en el producto.

RECOMENDACIONES

Los objetivos generales de este trabajo fueron alcanzados, pero durante su desarrollo, han surgido ideas que sería recomendable tener en cuenta para su futuro perfeccionamiento:

- Utilizar en los proyectos que aplican la metodología MA-GMPR-UR2 el nuevo procedimiento propuesto para las pruebas de calidad en dicha metodología.
- Brindar cursos de metodologías ágiles a estudiantes y profesores de la Universidad de las Ciencias Informáticas.
- Continuar documentando el proceso de automatización de las pruebas propuestas.

REFERENCIA BIBLIOGRÁFICA

1. **Fabien Potencier, François Zaninotto.** *Symfony la guía definitiva.* 2008.

2. *REVISTA MEDICA HEREDIANA VOL.* **Cortijo, Luis Lozano.** Perú : s.n., 2003.

3. **Romero, Gladys Marsi Peñalver.** *MA-GMPR-UR2, Metodología ágil para proyectos de software libre.* Ciudad de la Habana : s.n., 2008. 7.

4. Agile Spain . [En línea] 7 de 11 de 2005. http://www.agile-spain.com/agilev2/principios_agiles.

5. **Borrell, Alberto.** homepages. [En línea] http://homepages.mty.itesm.mx/al1031357/XP.ppt.

6. **González, Ing. Guillermo.** opensolutions. [En línea] 14 de 4 de 2008. http://www.opensolutions.com.py/~ggonzalez/fpuna/is2/files/03_JUNIT.pdf.

7. **Pons, Yanet Fernández.** monografias. [En línea] http://www.monografias.com/trabajos36/pruebas-de-aceptacion/pruebas-de-aceptacion2.shtml.

8. cybertesis. [En línea] 2005. http://www.cybertesis.edu.pe/sisbib/2005/valdivia_ee/pdf/valdivia_ee-TH.4.pdf.

9. eclases. [En línea] 2004. http://eclases.tripod.com/id18.html.

10. **yawelak, ariel sabiguero.** jiap. [En línea] 2008. http://www.jiap.org.uy/jiap/JIAP2008/PDF/121-CES.pdf.

11. **adimen.** [En línea] 2008. http://adimen.si.ehu.es/~rigau/teaching/EHU/ISHAS/Curs2008-2009/Apunts/IS.14.pdf.

12. **González, Ing. Guillermo.** Open Solutions. [En línea] 14 de 4 de 2008. http://ww.opensolutions.com.py/~ggonzalez/fpuna/is2/files/03_JUNIT.pdf.

13. gabbos. [En línea] 21 de 9 de 2007. http://gabbos.blogspot.com/2007_09_01_archive.html.

14. **wikipedia.** wikipedia. [En línea] 20 de 5 de 2009. http://es.wikipedia.org/wiki/Desarrollo_%C3%A1gil_de_software.

15. wikipedia. [En línea] 7 de 5 de 2009. http://es.wikipedia.org/wiki/Historias_de_usuario.

BIBLIOGRAFÍA

1. **Alvarez, Raúl Velázquez.** *Propuesta de una estrategia de Aseguramiento de la Calidad para el proyecto Convenio Cuba-Venezuela para su segunda fase.* Ciudad de la Habana : s.n., 2008.

2. **Betancourt, Maria de los Angeles Rubalcaba.** *Medición de la calidad de Software durante el Proceso de Pruebas en el Proyecto Modernización del CICPC.* Ciudad de la Habana : s.n., 2008.

3. **Cortés, Oscar Hernando Guzmán.** *Aplicación práctica del diseño de pruebas de software a nivel de programación.* 2004.

4. **Lemus, Yudisleidys Peña.** *SIMETSE – SIstema de METricas para evaluar el Software Educativo.* Ciudad de la Habana : s.n., 2007.

5. **León, Rolando Alfredo Hernández.** *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA.* Ciudad de la Habana : Editorial Universitaria, 2002.

6. **M, Itzcoalt Alvarez.** SG Software Guru. [En línea] 26 de 11 de 2007. http://www.sg.com.mx/sg07/presentaciones/Mejora%20de%20procesos/SG07.P02.Scrum.pdf.

7. **Pérez, Yalina Jiménez.** *Procedimiento para el Aseguramiento de la Calidad en el Proyecto INSIGNE.* Ciudad de Habana : s.n., 2008.

8. PX Programacion Extrema. [En línea] 22 de 12 de 2007. http://www.programacionextrema.org/.

9. **Rodríguez, Eniel Corzo.** *Estrategia para el Aseguramiento de la Calidad del Proyecto Juegos CNeuro.* Ciudad de La Habana : s.n., 2008.

10. **Santiesteban, Yanet Coba.** *Propuesta de Plan de Aseguramiento de la Calidad del Proyecto Servicios Comunitarios.* Ciudad de la Habana : s.n., 2008.

11. **V, María José Roca.** *Pruebas de Integración de Productos: Un enfoque práctico.* Cali : s.n., 2005.

12. Wikipedia. [En línea] 16 de 5 de 2009. http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.

13. wikipedia. [En línea] 16 de 5 de 2009. http://es.wikipedia.org/wiki/Programaci%C3%B3n_extrema.

14. Wikipedia. [En línea] 21 de 5 de 2009. http://es.wikipedia.org/wiki/Scrum.

ANEXOS

A.1: Estructura de la Historia de Usuario

	Historia de Usuario
Número: *[Número de la Historia]*	Nombre Historia de Usuario: *[Número que identifica la Historia]*
Modificación de Historia de Usuario Número: *[Cantidad de modificaciones que se le ha realizado a la historia de usuario (de no tener modificaciones se pone ninguna, sino la cantidad de veces que ha sido modificada).]*	
Usuario: *[Programador responsable de su implementación]*	Iteración asignada: *[Que iteración se desarrollará (según su importancia)]*
Prioridad en el negocio: *[Puede Alto, medio, o bajo (según Programadores)]*	Puntos estimados: *[Tiempo en semanas que se le asignará (estimado)]*
Riesgo en desarrollo: *[Puede Alto, medio, o bajo (según Programadores)]*	Puntos reales: *[Tiempo real dedicado a la realización de la HU en semanas]*
Descripción: *[Breve descripción del proceso que define la historia.]*	
Observaciones: *[Alguna acotación importante de señalar acerca de la HU.]*	
Prototipo Interfaz: *[Imagen de cada una de las interfaces relacionadas con de HU.]*	

A.2: Diseños de Casos de Pruebas.

A.2.1: Diseño de caso de pruebas unitarias

<table>
<tr><th colspan="2"></th><th colspan="4">Caso de Prueba Unitaria</th></tr>
<tr><td colspan="6">Código del Caso de Prueba</td></tr>
<tr><td colspan="6">Iteración:</td></tr>
<tr><td colspan="6">Nombre de la funcionalidad a probar:</td></tr>
<tr><td>Ambiente:</td><td colspan="5">Contiene la información acerca de la configuración de hardware o software el cuál ejecutará el caso de prueba.</td></tr>
<tr><td colspan="6">Descripción de la prueba:</td></tr>
<tr><td colspan="6">Condiciones de ejecución de la prueba:</td></tr>
<tr><td>Número</td><td>Funcionalidad</td><td>Tipo de prueba</td><td>Recibe</td><td>Esperado</td><td>Resultado</td></tr>
<tr><td>Número de veces que se ha chequeado la prueba, en caso de ser la primera vez se pone cero</td><td>Ejemplo: Autenticar</td><td>“ok”
//en caso de ejecutarse manualmente especificar.</td><td>“drhernandez”,”pruebadefuego"</td><td>“ok”</td><td>“ok”</td></tr>
<tr><td colspan="6">Evaluación de la prueba: (una descripción de lo que el encargado de la prueba debería ver tras haber completado todos los pasos de la prueba)</td></tr>
</table>

A.2.2: Diseño de caso de prueba de Integración no incremental o Caso de Prueba de integración incremental

	Caso de Prueba de integración incremental o caso de Prueba de integración no incremental
Código Caso de Prueba:	
Nombre del encargado de la prueba	
Herramientas o facilidades utilizadas. *[en caso de hacerse de manera manual especificar el tiempo que se tardó en terminar la prueba]*	
Iteración	
Conjunto de HU a probar:	
Descripción de la Prueba:	
Valores de entrada de la prueba: *[Listado de las entradas con sus valores específicos que se necesita para realizar el proceso]*	
Condiciones de ejecución de la prueba: *[La condición que se debe dar en el sistema antes de ejecutar las funcionalidades]*	
Pasos para la ejecución de la Prueba: *[Se combinan todos los módulos por anticipado y se prueba todo el programa en conjunto]*	
Resultado Esperado: *[una descripción de lo que el encargado de la prueba debería ver tras haber completado todos los pasos de la prueba]*	
Defectos encontrados: *[Defectos encontrados enumerados]*	
Evaluación de la Prueba:	

A.2.3: Diseño de caso de prueba de Aceptación Alfa

	Caso de Prueba de aceptación alfa
Código Caso de Prueba: *[Inicial del proyecto-número de la HU a la que pertenece la prueba-número de la prueba.]*	Nombre Historia de Usuario *[Nombre de la HU a realizar prueba.]*
Nombre de la persona que realiza la prueba: *[Nombre y apellidos.]*	
Descripción de la Prueba: *[Pasos para realizar la prueba.]*	
Condiciones de Ejecución: *[Condiciones necesarias para poder realizar la prueba.]*	
Entrada / Pasos de ejecución: *[Serie de pasos necesarios para lograr la realización de la HU, y así realizar la prueba.]*	
Resultado Esperado: *[Que cumpla con las restricciones del producto.]*	
Evaluación de la Prueba: *[Satisfactoria o no satisfactoria.]*	

A.2.4: Diseño de caso de prueba de Aceptación Beta

	Caso de Prueba de Aceptación Beta
Código Caso de Prueba: *[Inicial del proyecto-número de la HU a la que pertenece la prueba-número de la prueba.]*	Nombre de las Historias de Usuario a probar *[Nombre de las HU a realizar prueba.]*
Código de la entrega *(el código me registra la cantidad de HU que yo le entrego y el nombre)*	
Nombre de la persona que realiza la prueba: [Nombre y apellidos.]	
Condiciones de ejecución: [Requisitos que se deben cumplir para realizar el caso de prueba.]	
Descripción de la prueba: [Pasos para realizar la prueba.]	
Entrada / Pasos de ejecución:	
Resultado Esperado: *[Que cumpla con las restricciones del producto, un resumen de lo que el cliente espera]*	
Comentarios:	
Grado de Satisfacción General: *[1…5]*	

Aspecto a Evaluar				**Valor de aceptación (1…5)**
Funcionalidad	Exactitud de la aplicación o Precisión.			
	Buen funcionamiento			
Usabilidad	Atractivo	Interfaz amigable	Diseño	
		Interfaz sobrecargada	Información	
			Gráficos	

	Manejo	Facilidad para operar	
		Dificultad para ser aprendido	
Fiabilidad	Madurez	Información de exigencias de hardware y software	
Otro aspecto			

Valor	Estado
1	Muy Malo
2	Malo
3	Regular
4	Bueno
5	Excelente

A.2.5: Diseño de caso de prueba de regresión

	Caso de Prueba de Regresión
Código Caso de Prueba:	
Nombre del encargado de la prueba	
Ambiente: (Contiene información acerca de la configuración del hardware o software en el cuál se ejecutará el caso de prueba.)	
Propósito de la prueba.	
Descripción de la Prueba:	
Tipo de error: *(Casi siempre son errores inesperados que se producen al producirse un cambio en el programa.)*	
Localización del error: *(En que parte del sistema se encuentra localizado el error encontrado)*	
Acción correctiva a usar:	
Resultado Esperado: *(una descripción de lo que el encargado de la prueba debería ver tras haber completado todos los pasos de la prueba)*	
Evaluación de la Prueba:	

Tipo de error	
1	Los cambios introducen nuevos errores.
2	Los cambios revelan errores previos.
3	Los cambios vinculan alguna otra parte del programa e introducen errores en ella.
4	Los cambios realizados con respecto a nuevas funcionalidades introducen errores en nuevas funcionalidades realizadas en la versión actual.
5	Los cambios realizados con respecto a nuevas

	funcionalidades introducen errores en funcionalidad existente de previas versiones.
6	Otro. //en caso de ser este realizar una descripción del mismo

A.3: Métodos para las pruebas unitarias del objeto lime_test de Symfony y de Zend Frameworks

A.3.1: Métodos para las pruebas unitarias del objeto lime_test de Symfony

Método	Descripción
diag($mensaje)	Esta función muestra un comentario, pero no ejecuta ninguna prueba.
ok($prueba, $mensaje)	En esta función si la condición que se indica es true, la prueba tiene éxito.
is($valor1, $valor2, $mensaje)	Esta función compara 2 valores y la prueba pasa si los 2 son iguales.
isnt($valor1, $valor2, $mensaje)	Esta función compara 2 valores y la prueba pasa si no son iguales.
like($cadena, $expresionRegular, $mensaje)	Prueba que una cadena cumpla con el patrón de una expresión regular.
unlike($cadena, $expresionRegular, $mensaje)	Prueba que una cadena no cumpla con el patrón de una expresión regular.
cmp_ok($valor1, $operador, $valor2, $mensaje)	Compara 2 valores mediante el operador que se indica.
isa_ok($variable, $tipo, $mensaje)	Comprueba si la variable que se le pasa es del tipo que se indica.
isa_ok($objeto, $clase, $mensaje)	Comprueba si el objeto que se le pasa es de la clase que se indica.
can_ok($objeto, $metodo,	Comprueba si el objeto que se le pasa dispone del método que

$mensaje)	se indica.
is_deeply($array1, $array2, $mensaje)	Comprueba que 2 arrays tengan los mismos valores.
include_ok($archivo, $mensaje)	Valida que un archivo existe y que ha sido incluido correctamente.
fail()	Provoca que la prueba siempre falle (es útil para las excepciones).
pass()	Provoca que la prueba siempre se pase (es útil para las excepciones).
skip($mensaje, $numeroPruebas)	Cuenta como si fueran $numeroPruebas pruebas (es útil para las pruebas condicionales).
todo()	Cuenta como si fuera 1 prueba (es útil para las pruebas que todavía no se han escrito).

A.3.2: Ejemplos de Sentencias para pruebas unitarias de Zend Frameworks

Sentencia	**Descripción**
assertEquals (bolean excepted, bolean actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (byte expected, byte actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (char expected, char actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (double expected, double actual, double delta)	Verifica que los valores proporcionados sean iguales tomando en cuenta la tolerancia indicada por el parámetro delta.
assertEquals (float expected,	Verifica que los valores proporcionados sean

float actual, flota delta)	iguales tomando en cuenta la tolerancia indicada por el parámetro delta.
assertEquals (int expected, int actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (long expected, long actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (java.lang.Object expected, java.lang.Object actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (short expected, short actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (java.lang.String message, bolean expected, bolean actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.String message, byte expected, byte actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.String message, char expected, char actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.string message, double expected, double actual, double delta)	Verificar que los valores proporcionados sean iguales dentro de la tolerancia especificada por el parámetro delta. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.string message, float expected, float actual, float delta)	Verificar que los valores proporcionados sean iguales dentro de la tolerancia especificada por el parámetro delta. Si no lo son, envía el mensaje

	indicado por el parámetro message.
assertEquals (java.lang.String message, int expected, int actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.String message, long excepted, long actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.String message, java.lang.Object excepted, java.lang.Object actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.String message, short excepted, short actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertEquals (java.lang.String excepted, java.lang.String actual)	Verifica que los valores proporcionados sean iguales.
assertEquals (java.lang.String message, java.lang.String expected, java.lang.String actual)	Verifica que los valores proporcionados sean iguales. Si no lo son, envía el mensaje indicado por el parámetro message.
assertFalse (bolean condition)	Verifica que el objeto bolean seleccionado sea falso.
assertFalse (java.lang.String message, bolean condition)	Verifica que el objeto bolean seleccionado sea falso.
assertNotNull (java.lang.Object	Verifica que el objeto proporcionado no sea null.

object)	
assertNotNull (java.lang.String message, java.lang.Object object)	Verifica que el objeto proporcionado no sea null y envía un mensaje en caso de que sea cierto.
assertNotSame (java.lang.Object expected, java.lang.Object actual)	Verifica que los objetos proporcionados no son los mismos.
assertNotSame (java.lang.String message, java.lang.Object expected, java.lang.Object actual)	Verifica que los objetos proporcionados no son los mismos.
assertNull (java.lang.Object object)	Verifica que el objeto proporcionado es null.
assertNull (java.lang.String message, java.lang.Object object)	Verifica que el objeto proporcionado es null enviando un mensaje en caso de que no se cumpla la condición.
assertSame (java.lang.Object expected, java.lang.Object actual)	Verifica que los objetos proporcionados son los mismos.
assertSame (java.lang.String message, java.lang.Object expected, java.lang.Object actual)	Verifica que los objetos proporcionados son los mismos enviando un mensaje en caso de que no se cumpla la condición.
assertTrue (bolean condition)	Verifica que el parámetro proporcionado true.
assertTrue (java.lang.String message, boolean condition)	Verifica que el parámetro proporcionado sea true enviando un mensaje de error en caso de que no sea así.

A.4: Documento Garantía de Calidad

Iteración […]

Casos de Pruebas de Unitarias

//Casos de pruebas unitarias realizadas, esto es de las pruebas de subsistemas y de sistema completo...

Casos de Pruebas de Integración

// Casos de pruebas de prueba de integración realizadas, esto es de las pruebas de subsistemas y de sistema completo...

Casos de Pruebas de aceptación

// Casos de pruebas de aceptación realizadas...

Casos de Pruebas de regresión

// Casos de pruebas de regresión realizadas...

Resultado de los casos de prueba

//Un breve resumen sobre los resultados de las pruebas en la iteración

A.5: Entrevista a realizara a los gerentes y clientes de los proyectos NovaDesk y dotproject SXP

1. ¿Cómo se desarrolla el proceso de pruebas en el proyecto?
2. ¿De que manera se realizan las pruebas unitarias al sistema?
3. ¿Cómo es el desarrollo actual de las pruebas en el proyecto?
4. ¿Los casos de prueba de aceptación se encuentran documentados?
5. ¿Existe un registro de errores detectados durante el desarrollo de las iteraciones del proyecto?
6. ¿Quiénes son los usuarios finales que van interactuar con la aplicación que se está desarrollando?
7. ¿Qué garantía hay de que los futuros usuarios tengan el mismo grado de satisfacción que el cliente que interactúa con el negocio con respecto a sistema que se desarrolla?

A.3.6: Encuesta para la validación de la propuesta mediante la técnica de panel de expertos

1) ¿Considera usted que el procedimiento propuesto está a la altura de las necesidades de la metodología MA-GMPR-UR2?

Si___ No___ ¿Por qué?

2) ¿Considera usted que el desarrollo de las pruebas propuestas es lo suficientemente factible a las necesidades de los proyectos productivos que utilicen la metodología MA-GMPR-UR2?

Si___ No___ Si cree necesario que alguna no va acorde con las necesidades, menciónelo y explique brevemente.

3) En una escala del 1 al 5 confiera una evaluación a la propuesta según los siguientes criterios:

___Satisfacción a las necesidades de la metodología MA-GMPR-UR2.

___Adaptabilidad a los proyectos productivos.

___ Mejora del proceso de prueba.

___Posibilidad de aplicación.

A.7: Información de confianza de los expertos

Experto	Vinculado a Proyectos Productivos	Graduado de	Categoría Científica	Eventos científicos	Experiencias en la pruebas de calidad de software	Responsabilidad que ocupa
Experto 1	Hace 15 años	Sistemas Automatizados de Dirección	Máster en Ciencias. Máster Ejecutivo de la EOI.	Varios	10 años	Director de Desarrollo y Calidad Correos de Cuba.
Experto 2	Sí	Ing. en Ciencias Informáticas	-	-	3 años de experiencia en el equipo de calidad.	Responsable del Departamento de Calidad del Centro de Identificación y Seguridad Digital.
Experto 3	Calidad de Software	Ingeniero Informático	-	Uciencia Forum de Ciencia y Técnica Exposición BTJ Universidad 2010	Se desempeñó como especialista de pruebas vinculada al proyecto Identidad durante 1 año	Jefe Departamento de Ingeniería de Software y Práctica Profesional

Experto 4	-	Ingeniera Informática	-	Uciencia Informática 2009 Artículo en Revista Cubana de Informática	4 años	Especialista general de Calidad
Experto 5	-	Ingeniería Informática	-	-	2 años	VDPI facultad 1
Experto 6	Si	Ingeniería Informática		UCI 2008 Forum de Ciencia y Técnica 2008	4 años	Líder de la Línea residencia del ERP Gestión Universitaria
Experto 7	Especialista de CALISOFT	Ingeniero en Ciencias Informáticas	-	UCIENCIA Fórum de Ciencia y Técnica. Universidad 2010	1 año como graduado y 3 años como integrante de un proyecto de calidad	Especialista

GLOSARIO DE TÉRMINOS

- ✓ SXP: Scrum y Extreme Programming
- ✓ UCI: Universidad de las Ciencias Informáticas
- ✓ TDD: Desarrollo Dirigido por pruebas
- ✓ CU --- Caso de uso
- ✓ HU ---- Historia de Usuario
- ✓ MVC ---- Modelo Vista Controlador
- ✓ RUP ---- Proceso Unificado de Desarrollo de Software (Rational Unified Process)
- ✓ XP---- Extreme Programming
- ✓ LRP ---- Lista de Reserva del Producto
- ✓ IGSW ---Ingeniería de Software

1