TEST CASES IN SOFTWARE TEST

LOS CASOS DE PRUEBA EN LA PRUEBA DEL SOFTWARE

José Luis Aristegui O.
Grupo IngeSoft, Chile.
Ingesoft@techie.com

(Artículo de REFLEXIÓN) (Recibido el 5 de enero de 2010. Aprobado el 12 de abril de 2010)

Abstract *— In the place of traditional principle of project management a strategic management philosophy is emerging fast in which writing better test cases also receive the widespread attention of all those interested in software project management and software testing. In the current scenario managing software is an important task in an IT industry. Not only managing IT project, but also it is need to develop quality software product for the customer. It includes the number of tasks and phases of the software project development. Testing is one of the phases, which is most important in project management. In software testing writing test cases is very important. So it is necessary to study how to write better test cases.*

Keywords*: quality test cases, software testing, test cases.*

Resumen — En el campo de la gestión tradicional de proyectos de software, surgió hace poco una filosofía estratégica que se centra en mejorar el diseño de los casos de prueba, y que llamó la atención generalizada de los interesados en la gestión de proyectos y en la prueba del software. Actualmente, la gestión de proyectos de software es una de las tareas más importantes en la industria de las tecnologías de la información, y más aún si el objetivo es desarrollar productos de calidad. En esa gestión, la prueba es una de las fases más importantes, y en ésta, lo que requiere más cuidado y dedicación es el diseño de los casos de prueba, por lo que es necesario estudiar cómo diseñarlos y escribirlos mejor.

Palabras clave: calidad de los casos de prueba, casos de prueba, prueba del software.

INTRODUCCIÓN

La gestión de proyectos es un complejo sistema de procedimientos de gestión, prácticas, tecnologías y conocimientos, en el que es necesaria la experiencia para gestionarlos con éxito. La gestión de proyectos de software es una actividad lineal en la Ingeniería de Software. Se inicia antes que cualquier actividad técnica comience y continúa durante todas las etapas de desarrollo hasta el mantenimiento.

Figura 1. La administración de proyectos
(Jalote, 2002)

Gestionar el desarrollo de software como un proyecto es muy importante; se trata de integrar personas —desarrolladores, clientes—, problemas —requisitos del cliente—, y procesos —metodología para encontrar los requisitos—; es decir, integrar las tres P: Personas, Procesos y Problemas, como se observa en la Figura 1 (Jalote, 2002).

Para desarrollar productos software de calidad, la prueba es una de las tareas más importantes y, cuando se aplica linealmente en el ciclo de vida del producto, desempeña un papel crucial en la gestión de proyectos. Las pruebas evalúan el producto para determinar que cumple con el objetivo previsto, por lo que es necesario diseñar un plan de pruebas que se adapte y sea coherente con la metodología de desarrollo, que proporcione un enfoque de fácil acceso a la estructura para verificar los requisitos y cuantificar su rendimiento, y que identifique las diferencias entre los resultados previstos y los reales —errores o fallas—; es el proceso por medio del cual se evalúa la correcta interpretación y aplicación de los requisitos especificados. Un buen plan de pruebas es el conocido como PDCA (Yongkui and Guofeng, 2009), que contempla las siguientes actividades:

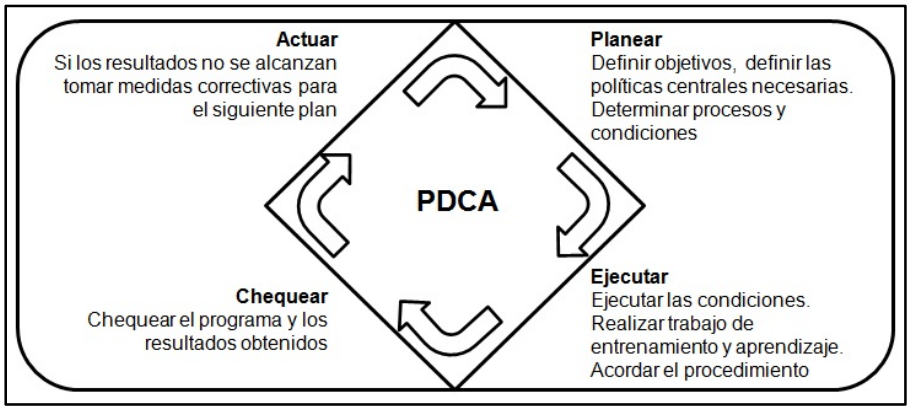

Figura 2. Ciclo de la prueba del software (Yongkui and Guofeng, 2009)

Este artículo detalla cómo evitar los contratiempos que se originan cuando los casos de prueba se diseñan pobremente, y describe cómo mejorar ese diseño para incrementar la productividad, la facilidad de uso, la fiabilidad de la programación y la gestión de valor. Además, se busca explicar qué son y para qué sirven los casos de prueba, así como la lista de estándares que se utilizan para identificar áreas de riesgo y mejorarlos para aplicaciones futuras. Se desarrolla un caso de estudio que demuestra cómo utilizar los casos de prueba para mejorar la capacidad de la prueba y la productividad, resolver los problemas más comunes en calidad y cómo aprovechar las ventajas de los casos de prueba, que se pueden poner en práctica en la industria del software.

ESTADO ACTUAL DE LA PRÁCTICA DEL DESARROLLO DE SOFTWARE

Toda persona, alguna vez, ha sufrido algún error del software, sea en una factura liquidada indebidamente o la pérdida de todo un día de trabajo; estos problemas se deben a la complejidad misma del software, que dificulta el desarrollo de sistemas e incrementa la probabilidad de que existan errores aun luego de finalizado y entregado al cliente.

Actualmente, la capacidad de los ingenieros de software para medir la fiabilidad de sus productos es inferior a la necesaria en el enfoque de la calidad (Gibbs, 1994), por lo que es deseable que estos ingenieros puedan demostrar matemáticamente la correctitud de sus programas, de la misma forma que otras ramas de la ingeniería lo pueden hacer. Otros ingenieros pueden recurrir al análisis matemático para conocer de antemano el comportamiento de sus productos en el mundo real, para descubrir sus errores antes que estén operativos. No obstante, las matemáticas tradicionales utilizadas para describir sistemas físicos no tienen aplicabilidad en el universo de los computadores, en el que se debe recurrir a la matemática discreta, un área reciente, poco investigada, y que gobierna el campo de los sistemas software (Roventa and Spircu, 2008).

Es debido a esas imposibilidades que los ingenieros de software no pueden aplicar los métodos matemáticos rigurosos a sus productos, y deben recurrir a métodos de verificación empírica en los que hacen funcionar los programas y observan su comportamiento directamente, para luego depurarlos cada vez que aparece un error; de esta forma la fiabilidad de los productos se va incrementando a lo largo del proceso. Estos métodos no garantizan la ausencia de errores, y pueden existir errores en otros componentes del programa que no se ejecutan en ese momento. Por lo tanto, la recomendación es que el producto software se evalúe de forma paralela a su desarrollo, en un proceso de evaluación/comprobación de los diferentes productos en cada fase del ciclo de vida, y en el que participen desarrolladores y clientes.

Se concluye entonces que el logro de programas perfectos es por el momento una meta inalcanzable. "*Existe, actualmente, la imposibilidad práctica de conseguir software totalmente libre de defectos*" (Littlewood and Strigini, 1993), por lo que se debe aceptar que, debido a las actuales limitaciones en lo relacionado con el desarrollo de sistemas software, esta práctica debe investigarse más y más cada día; de hecho, existen autores que sugieren

que "[…] *debido a la entidad no física del software, los errores en los programas son inherentes a su naturaleza*" (Huang et al., 1994), y de hecho, hasta el software de más alto factor crítico contiene errores remanentes. En su investigación, Edward Adams analizó el número de errores en una base de datos, que se suponía de cobertura mundial, en un equivalente a miles de años de uso sobre un sistema de cómputo. Descubrió que uno de cada tres errores del programa llegan a producir un fallo tan sólo una vez cada 5.000 años. Claro está que emplear tiempo para detectar los errores que se producirán más allá de 75 años es inaceptable (Adams, 1984).

LOS CASOS DE PRUEBA EN LA INGENIERÍA DE SOFTWARE

Para desarrollar software de calidad y libre de errores, el plan de pruebas y los casos de prueba son muy importantes. El *Software Test Plan —STP—* se diseña para determinar el ambiente de aplicación de los recursos y el calendario de las actividades de las pruebas, se debe identificar el dominio y sus características a probar, lo mismo que el tipo de pruebas a realizar. Un caso de prueba bien diseñado tiene gran posibilidad de llegar a resultados más fiables y eficientes, mejorar el rendimiento del sistema, y reducir los costos en tres categorías: 1) productividad —menos tiempo para escribir y mantener los casos—; 2) capacidad de prueba —menos tiempo para ejecutarlos—; y 3) programar la fiabilidad —estimaciones más fiables y efectivas— (Perry, 1995).

La prueba del software consta de tres pasos: el entorno de la prueba, desarrollar y ejecutar *scripts*, y analizar los resultados (IEEE, 2008); el plan de pruebas "*describe el alcance y enfoque de las actividades de pruebas previstas, e identifica las características a ser probadas*" (Elaine and Vocolos, 2000); y el diseño de las pruebas "*especifica los detalles del método de prueba para una característica del software e identifica las pruebas correspondientes*" (IEEE, 2008-1). El estándar 829 de IEEE (IEEE, 2008-2) recomienda que un plan de pruebas debe incluir:

- Una lista de características y sus combinaciones a probar
- Una declaración general de enfoque para cada característica o combinación de características
- Identificación de la prueba de diseño asociada con cada una de las características y sus combinaciones.

El proceso de escribir casos de prueba y establecer su estándar es un logro especial muy dinámico, y es necesario que se enseñe, aplique, controle, mida y mejore continuamente.

Componentes de los casos de prueba

Un caso de prueba es un conjunto de acciones con resultados y salidas previstas basadas en los requisitos de especificación del sistema; sus componentes son:

1. Propósito: de la prueba o descripción del requisito que se está probando
2. Método: o forma como se probará
3. Versión: o configuración de la prueba, versión de la aplicación en prueba, el hardware, el software, el sistema operativo, los archivos de datos, entre otros
4. Resultados: acciones y resultados esperados o entradas y salidas
5. Documentación: de la prueba y sus anexos.

En cada nivel de la prueba, estos componentes deben probarse utilizando casos de prueba para pruebas de unidad, pruebas de integración, pruebas del sistema, pruebas Alpha y Beta,…, además, son útiles para las pruebas de rendimiento, pruebas funcionales y pruebas estructurales.

Factores de calidad de los casos de prueba

La calidad es un conjunto de métricas estándar o listas de control, y representa lo que los clientes buscan en un producto. Un caso de prueba debe cumplir con los siguientes factores de calidad:

1. Correcto. Ser apropiado para los probadores y el entorno. Si teóricamente es razonable, pero exige algo que ninguno de los probadores tiene, se caerá por su propio peso.
2. Exacto. Demostrar que su descripción se puede probar.
3. Económico. Tener sólo los pasos o los campos necesarios para su propósito.
4. Confiable y repetible. Ser un experimento controlado con el que se obtiene el mismo resultado cada vez que se ejecute, sin importa qué se pruebe.
5. Rastreable. Saber qué requisitos del caso de uso se prueban.

6. Medible. Este es un ejercicio muy útil para quienes escriben pruebas, para verificar constantemente dónde están, si pierden alguno de los elementos, o si no se cumple un estándar.

Formato de los casos de prueba

1. *Paso a paso*. Este formato se utiliza en:
   - La mayoría de las reglas de procesamiento
   - Casos de prueba únicos y diferentes
   - Interfaces GUI
   - Escenarios de movimiento en interfaces diferentes
   - Entradas y salidas complicadas para representar en matrices.

2. *Matrices*. Sus usos más productivos son:
   - Formularios con información muy variada, mismos campos, valores y archivos de entrada diferentes
   - Mismas entradas, diferentes plataformas, navegadores y configuraciones
   - Pantallas basadas en caracteres
   - Entradas y salidas con mejor representación matricial.

3. *Scripts automatizados*. La decisión de utilizar software para automatizar las pruebas depende de la organización y del proyecto que se esté probando. Existen algunas cuestiones técnicas que deben concretarse y que varían de una herramienta a otra. El beneficio real de automatizar las pruebas se obtiene en la fase de mantenimiento del ciclo de vida del software; en ese momento, los *scripts* se pueden ejecutar repetidamente, incluso sin supervisión, y el ahorro en tiempo y dinero es sustancial (Moller and Paulish, 1993).

Un caso de prueba paso a paso tiende a ser más verbal, y el de matrices más numérico. A menudo, la ruta más productiva es utilizarlos todos. Los dos primeros se utilizan para las pruebas unitarias, de integración y del sistema; y el automatizado, para pruebas de regresión (Voas, 1993).

Tabla 1. Mitos y realidades de los casos de prueba

1	Mito	Los casos de prueba paso a paso toman mucho tiempo para escribirse. No lo podemos permitir.
	Realidad	Puede o no que tomen más tiempo para escribirse, pero su detalle los hace resistentes y fáciles de mantener; además, son necesarios para probar adecuadamente algunas de las funciones.
2	Mito	Una matriz es siempre la mejor opción. Hagámosla trabajar.
	Realidad	Un problema persistente es armar una matriz con la información adecuada de la configuración. Frecuentemente se omite dicha información, o peor aún, si las configuraciones o clases de entrada son diferentes no se pueden forzar dentro de una matriz como grupo similar, ya que no se han probado todos.
3	Mitos	La alta tecnología es la mejor. Si es posible automatizar los casos de prueba, se debe hacer.
	Realidad	La decisión de utilizar pruebas automatizadas debe basarse en muchos factores.
4	Mito	No tenemos tiempo para escribir los casos de prueba manuales. Vamos a automatizarlos.
	Realidad	Automatizar los casos de prueba toma más tiempo que los otros dos tipos.

MEJORAMIENTO DE LOS CASOS DE PRUEBA

Comprobabilidad de los Casos de Prueba

En la prueba es fácil de probar, con precisión, lo que significa que si el probador sigue las instrucciones, el resultado de aprobado o fallido será correcto. Se puede medir fácilmente por medio del tiempo que se tarda en ejecutar la prueba, y si el probador tiene que buscar o no aclaraciones en el proceso de prueba.

- Lenguaje para mejorar la comprobabilidad. Los pasos de los casos de prueba deben ser escritos en forma activa. El probador debe saber qué hacer, y cómo hacerlo. Por ejemplo, navegar en la página de la tienda *online* y preparar la lista de lo que va a comprar, para comparar los precios y la variedad con los datos disponibles. Hacer clic en <*Submit*>, etc., puede hacerse más rápido mediante la creación de campos estructurados para que el probador registre las entradas que se verificarán y comprobarán posteriormente.
- Controlar longitud para mejorar la comprobabilidad. Es necesario tener en cuenta la longitud de los casos de prueba para saber cuán compleja y precisa es la prueba. Un buen caso de prueba debe tener entre 8 y 16 pasos —en el método paso a paso—, a menos que el probador no

pueda guardar su trabajo. Existen varias ventajas en mantener los casos de prueba cortos: se requiere menos tiempo y hay menos posibilidades de cometer errores, de necesitar ayuda o de alguna pérdida de datos. Con base en la longitud de los casos de prueba es posible estimar con precisión el tiempo y el esfuerzo que se debe invertir en la prueba, lo mismo que sus resultados. En los casos de prueba de matriz, una buena longitud oscila entre 18 y 20 minutos para la prueba. Mientras que la longitud de un *script* automatizado no es cuestión que interese para la ejecución de la prueba, ya que ésta se ejecuta en fracciones de segundo; la cuestión es administrar los contenidos, el mantenimiento y el seguimiento de los defectos.

- Pros y contras de los casos acumulativos. Estos casos son los que dependen de otros previamente ejecutados. El objetivo del probador es mantener la prueba de forma autónoma hasta que le sea posible, ya que esto le da mayor flexibilidad en su programación, y reduce el costo y el tiempo de mantenimiento. Si el caso de prueba depende de otro, entonces podrá referenciarlos y en tal caso la prueba será acumulativa. Siempre que sea necesario, debe ofrecer una alternativa a una prueba anterior; esto implica que se pueden utilizar los datos creados en una prueba anterior, pero que también podría utilizar otros datos. En general, se deben mantener las referencias a otras pruebas lo más genéricas y compatibles posible, y no referirse a los casos de prueba solamente por su número (Pressman, 2004).

Plantillas para mejorar la productividad

La plantilla para casos de prueba es un formulario con campos marcados o símbolos, y sirve para mejorar los casos de prueba paso a paso; sirve para darle orden a los casos de prueba, ya que evita la indeseable página en blanco, y se basa en normas. Es posible observar el proceso impreso y ayuda a los probadores a encontrar información.

Tabla 2. Plantilla para casos de prueba

Proyecto No.: Nombre del Proyecto:	Página No.:
Caso No.: Nombre del Caso:	Ejecución No.: Nombre: Estado de la prueba:
Marca/Subsistema/Módulo/Nivel/Función/Código de la Unidad bajo prueba:	Requisito No.: Nombre:
Escrito por: Fecha:	Ejecutado por: Fecha:
Descripción del caso de prueba (propósito y método):	
Configuración de la prueba para (H/W, S/W, N/W, datos, pre-requisitos de prueba, seguridad y tiempo):	

Paso	Acción	Resultados esperados	Pasado/Fallido
1			
2			
…			

También se recomiendan plantillas para las matrices (Watts, 1989):

Tabla 3. Plantilla para matrices

Proyecto No.:			Nombre del proyecto:					Página:		
Nombre de la prueba:			Construcción No.: Fecha de ejecución: Nombre ejecutor:					Ejecución No.:		
Escrito por:			Fecha:					Requisito No.:		
Descripción del caso de prueba (propósito y método):										
Configuración de la prueba:										
Pasado/ Fallido	Usuario	Visualiza	Edición	Adición	Borrado	Reconst.	Auditar	Report.	Seguir	Result.
	1									
	2									
	3									
	…									

- Clonar mejora la productividad. Significa modelar un caso de prueba en otro. Un caso de prueba es un buen candidato para clonar si se ajusta a la necesidad de un caso paso a paso y tiene variables que puedan ser fácilmente sustituidas. Investigar en torno del proyecto las oportunidades de clonación, como los casos de otras personas, manuales de usuario o tutoriales *help desk*, que pueden estar buscando un intercambio para los casos de prueba. Las matrices también pueden ser clonadas, sobre todo si la sección de configuración es la misma. Las variables pueden cambiar en los nombres de campo y sus valores.
- Administrar las pruebas mejora la productividad. Un software diseñado para soportar pruebas de auditoría incrementa la productividad al escribir los casos de prueba, y tiene ventajas sobre los procesadores de texto, base de datos o software de hoja de cálculo. Administrar la prueba del software hace que escribir y diseñar los casos de prueba sea más fácil, facilita la clonación de los casos y sus pasos, hace fácil agregarlos, moverlos y eliminarlos, los numera automáticamente y renumera e imprime las pruebas con facilidad desde las plantillas (Watts, 1995).

CASO DE ESTUDIO

Los probadores eran usuarios de un negocio con poca confianza en el software; estaban deseosos de probar, y después de pasar mucho tiempo tratando de averiguar qué hacer con las pruebas, casi tiraban la toalla. El analista de las pruebas observó los casos de prueba y comenzó un programa de formación y orientación al equipo; convencido de que los casos de prueba tenían gran utilidad, le dio a los probadores una lista de control e hizo que el grupo la usara para escribir los casos del siguiente módulo. Cada uno comenzó a producir casos que cumplían los estándares. La primera semana la escritura fue lenta, pero luego de dos meses todos lograron alcanzar el objetivo de productividad. En el ciclo de prueba siguiente, los casos eran más cortos, cada uno con un objetivo claro y criterios precisos para aprobar o rechazar.

El administrador de las pruebas mantenía indicadores acerca de los cambios. Los analistas gastaban entre cuatro y cinco horas diarias solucionando problemas de casos deficientes con los probadores. Además del tiempo dedicado a los probadores, algunos analistas gastaban una o dos horas tratando de arreglar los casos en cada fase, en lugar de realizar su trabajo regular: escribir casos de prueba para el siguiente módulo.

Después de la capacitación y del establecimiento de estándares, las métricas para el próximo ciclo de prueba se veían mucho mejor. Ninguno de los analistas pasaba más de una hora al día ayudando a los probadores. A pesar de que había más pruebas, los casos de prueba eran más cortos debido a los estándares; los probadores los aplicaban en veinticinco minutos, y a menudo las pruebas terminaban un día antes. Al final del proyecto, los analistas y probadores recibían reconocimiento por ofrecer una versión completa y a tiempo del producto, e inclusive por ahorrarse un mes en el ciclo de vida.

CONCLUSIONES

- La actividad más importante para proteger el valor de los casos de prueba es mantener los que son comprobables. Deben mantenerse después de cada ciclo de prueba, ya que los probadores encuentran errores en los casos de prueba, lo mismo que en el software. Cuando se programan las pruebas, se debe asignar tiempo para que el analista o escritor diseñe los casos, mientras los desarrolladores corrigen los errores en la aplicación.
- El director del proyecto o el administrador de las pruebas, debe encargarse de administrar la configuración para proteger el valor de los casos de prueba, así como de los estándares.
- El proceso de enseñar buenas prácticas de escritura y de establecer estándares para los casos de prueba es un logro en sí mismo; pero no pueden ser estáticas: deben enseñarse de forma dinámica, aplicadas, controladas, medidas y mejoradas.
- Este documento ha cubierto brevemente cuáles son los procesos y estándares de calidad y cómo se aplican a todo tipo de casos de prueba, cómo usarlos para mejorar la comprobabilidad y productividad de la prueba, cómo resolver los problemas comunes de calidad en los

casos de prueba, y cómo proteger el activo de los casos de prueba.

- Para que un caso de prueba se considere de buena calidad se recomienda la siguiente lista de chequeo (Dustin, 2003):

Tabla 4. Lista de comprobación para la calidad de un caso de prueba

Atributo	Lista de chequeo	S/N
Calidad	Correcto. Es apropiado para los probadores y el entorno	
	Exacto. Su descripción se puede probar	
	Económico. Tiene sólo los pasos o los campos necesarios para su propósito	
	Confiable y repetible. Se obtiene el mismo resultado sin importa qué se pruebe	
	Rastreable. Se sabe qué requisito se prueba	
	Medible. Retorna al estado de la prueba sin valores en su estado	
Estructura y capacidad de prueba	Tiene nombre y número	
	Tiene un propósito declarado que incluye qué requisito se está probando	
	Tiene una descripción del método de prueba	
	Especifica la información de configuración —entorno, datos, pre-requisitos de prueba, seguridad—	
	Tiene acciones y resultados esperados	
	Guarda el estado de las pruebas, como informes o capturas de pantalla	
	Mantiene el entorno de pruebas limpio	
	No supera los 15 pasos	
	La matriz no demora más de 20 minutos para probarse	
	El *script* automatizado tiene propósitos, entradas y resultados esperados	
	La configuración ofrece alternativas a los pre-requisitos de la prueba cuando es posible	
	El escenario de aplicación se relaciona con otras pruebas	
Administración de la configuración	Emplea convenciones de nomenclatura y numeración	
	Guarda en formatos especificados los tipos de archivo	
	Su versión coincide con el software bajo prueba	
	Incluye objetos de prueba necesarios para el caso, tales como bases de datos	
	Almacena con acceso controlado	
	Realiza copias de seguridad en red	
	Archiva por fuera del sitio	

REFERENCIAS

1. Adams, E. (1984). Optimizing Preventive Service of Software Products. IBM Research Journal, Vol. 28, No. 1, pp. 2-14.
2. Dustin, E. (2003). Effective software testing: 50 Specific Ways to Improve Your Testing. New York: Addison-Wesley Professional.
3. Elaine, J. and Vocolos F. I. (2000). Experience with performance testing of software systems: Issues, approach and case study. IEEE transactions on software engineering, Vol. 26, No. 12, pp. 1147-1156.
4. Gibbs, W. W. (1994). Software's Chronic Crisis. Scientific American, No. 218, pp. 72-81.
5. Huang, Y., Jalote P. and Kintala C. (1994). Two Techniques for Transient Software Error Recovery. Lecture Notes in Computer Science, Vol. 774, pp. 159-170.
6. IEEE Computer Society. (2008). IEEE Standard for Software and System Test Documentation. IEEE Standard 829, Section 3 "Definition", pp. 8.
7. IEEE Computer Society. (2008-1). IEEE Standard for Software and System Test Documentation. IEEE Standard 829, Section 10 "Level test design", pp. 50.
8. IEEE Computer Society. (2008-2). IEEE Standard for Software and System Test Documentation. IEEE Standard 829.
9. Jalote, P. (2002). Software project management in practice. Boston: Pearson Education.
10. Littlewood, B. and Strigini L. (1993). The Risks of Software. Scientific American, Vol. 268, No. 1, pp. 62-75.
11. Moller, K. and Paulish D. (1993). Software matrics: A practitioner guide to improve product development. London: Champnan and Hall.
12. Perry, W. (1995). Effective methods for software testing. New York: John Wiley.
13. Pressman, R. S. (2004). Software engineering: A practitioner approach. New York: Mc-Graw Hill International.

14. Roventa, E. and Spircu T. (2008). The central role of discrete mathematics in the context of nformation technology and communications. Fuzzy Information Processing Society, NAFIPS'08. Annual Meeting of the North American, pp. 1-5.
15. Voas, J. (1999). Software quality's eight greatest myths. IEEE Software, Vol. 16, No. 5, pp. 118 -120.
16. Watts S. H. (1995). A discipline for software engineering. USA: Addison-Wesley publishing Company.
17. Watts, S. H. (1989). Managing the Software Process. USA: Addison-Wesley Publishing Company.
18. Yongkui, S. and Guofeng S. (2009). Process control system of roof disaster based on PDCA cycle. 16th International Conference on Industrial Engineering and Engineering Management, E&EM '09. Beijin, China, pp. 199-203.

Ω