Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño

Conference Paper · October 2015

CITATIONS
0

READS
1,980

1 author:

Leticia Davila-Nicanor
Universidad Autónoma del Estado de México (UAEM)
17 PUBLICATIONS **36** CITATIONS

Pruebas de Regresión Funcional Mediante el Uso de Patrones de Diseño.

Dávila-Nicanor Leticia[1], Omar Marín Guerrero[1], Irene Aguilar Juárez[2], Joel Ayala de la Vega[2]
[1] Centro Universitario UAEM Valle de México, Blvd. Universitario Predio San Javier s/n, Atizapán de Zaragoza, C.P. 01219, Estado de México, México
[2]Centro Universitario UAEM Texcoco, Av. Jardín Zumpango s/n, Fracc. El Tejocote, Texcoco Estado de México, México
ldavilan@uaemex.mx, iaguilarj@uaemex.mx, jayalad@uaemex.mx
***Área de participación**: Ingeniería de Software*

Resumen

En el desarrollo de los sistemas de software, la funcionalidad es la base para su especificación, diseño, mantenimiento y evolución. Cuando el sistema no cumple con la especificación de requerimientos o los requerimientos cambian por diversas causas, como nuevas metas de negocio, nuevas tecnologías o paradigmas de programación emergentes, mencionando algunas, se experimentan pérdidas con altos costos en la producción de software. El principal objetivo de la fase de pruebas es verificar la funcionalidad del sistema para obtener un mejor producto de software.

Los escenarios basados en regresión funcional tienen el objetivo de establecer en términos de Trazabilidad, la evaluación de la relación de los requerimientos funcionales con los componentes en la arquitectura del software. Este enfoque es muy costoso, en la actualidad pocos trabajos lo automatizan. En la presente propuesta se automatiza la generación de casos de prueba mediante la combinación de la matriz de trazabilidad de pruebas y el uso de los patrones de diseño para generar casos de pruebas de una forma dinámica, eficiente y confiable.

***Palabras clave**: Regresión funcional, Trazabilidad, Patrones de diseño*

Abstract

In the development of software systems, the functionality is the basis for specification, design, maintenance and evolution phase. When the system does not meet the requirements specification or requirements change for various reasons, such as new business goals, new technologies and emerging paradigms of programming, among others, losses are experienced with high costs in the software production. The goal of the testing phase is verify the functionality of the system in order to have a better system.

The scenarios based on functional regression aim to establish traceability in terms of the evaluation of the functional requirements with the components in the software architecture. This approach is very expensive, currently few jobs as automated. In this proposal the generation of test cases is automated by combining traceability matrix testing and the use of design patterns to generate test cases a dynamic, efficient and reliable manner.

Introducción.

Los sistemas de software están expuestos a cambios importantes durante su desarrollo, mantenimiento y evolución. Estas situaciones afectan la funcionalidad del sistema y la calidad que debe poseer en un ámbito productivo. El problema central de tener cambios de funcionalidad es que impactan de forma directa en la arquitectura del software y en su diseño. En el planteamiento del propósito del desarrollo de un sistema de software, se establecen los requerimientos funcionales y

no funcionales. En relación de los requerimientos funcionales es que se diseñan la arquitectura y los componentes del sistema de software. Cuando un sistema está en mantenimiento e incluso en evolución, tener clara la relación de los requerimientos con el diseño del sistema es central, porque es la base para extender o actualizar las funciones del sistema. Sin embargo es hasta la fase de pruebas cuando se evalúa la funcionalidad de un sistema.

En el presente trabajo se propone un novedoso esquema para automatizar la creación de casos de prueba tomando en cuenta la técnica de escenarios basados en regresión funcional. En base de los elementos estáticos de la matriz de trazabilidad de pruebas, se abstrae la funcionalidad de la aplicación mediante los patrones de diseño. Este enfoque permite construir estos casos en un contexto dinámico.

El desarrollo del artículo es de la siguiente forma, en la sección II es abordado el marco teórico y el trabajo relacionado. En el desarrollo de la sección III se describe el funcionamiento del sistema para la generación dinámica de casos de pruebas funcionales. En la sección IV se presenta la arquitectura y la forma en la que intervienen los patrones de diseño. Finalmente en la sección V, las conclusiones y el trabajo a futuro son descritos.

Marco teórico y trabajo relacionado.

El Instituto de Ingeniería de Software de la Universidad de Carnegie Mellon es uno de los principales órganos reguladores de los estándares y enfoques de la Ingeniería de Software a nivel internacional, en su *manual de medición de software* (Park, 1996) se marca con claridad que la mayoría de los fallos proviene de las primeras fases del proceso de desarrollo. La Figura 1 muestra valores que son indicadores en la relación de los problemas de origen y las fases del proceso de desarrollo de software. Es posible observar como un tratamiento inadecuado de los requerimientos afecta de forma directa al diseño, la codificación y en las pruebas; la formulación inadecuada de casos de prueba. En la fase productiva, cuando el sistema esta en *uso productivo*, se observa que problemáticas importante son la documentación del sistema, la inconsistencia del diseño y la codificación en relación con la funcionalidad del sistema. Cuando se realizan las pruebas de software, si los casos son insuficientes o tienen un alcance inadecuado, todo el proceso de pruebas se afecta negativamente y no se localizan la mayoría de los fallos potenciales. Con este enfoque, la Figura 2 establece las distribuciones del origen de los fallos en las primeras fases del proceso de desarrollo de software y su afección en porcentajes en las siguientes fases. Se puede observar en esta figura, que es muy costoso eliminar los fallos en fases finales. En este caso los valores se estiman en miles de marcos alemanes (KDM).

Trazabilidad de Requerimientos (Traceability).

La Trazabilidad es la propiedad directamente relacionada con la Funcionalidad del sistema y el diseño. Es aquella propiedad que permite enlazar las especificaciones de los requerimientos en relación con la arquitectura y la implementación del sistema. De acuerdo al *diccionario de estándares de métricas para producir software confiable* (IEEE Std. 830,1988), la Trazabilidad de los Requerimientos es una medida que ayuda a identificar los requerimientos tratados de forma inadecuada en relación de los requerimientos de origen. De acuerdo a la expresión 1, sus primitivas son:

$$M = R1/R2 * 100 \quad \ldots\ldots \quad (1)$$

M = Medida de trazabilidad.

R1 = Número de veces que requerimientos se proyecta en la arquitectura.

R2 = Número de veces que los requerimientos originales se proyectan en la arquitectura.

Para obtener la medida de trazabilidad, se obtiene un conjunto de mapeos de los requerimientos en la arquitectura de software en relación con los requerimientos originales. En este caso es la relación del número de cada requerimiento encontrado (R1) en contraste con el conteo de requerimientos originales proyectados en la arquitectura.

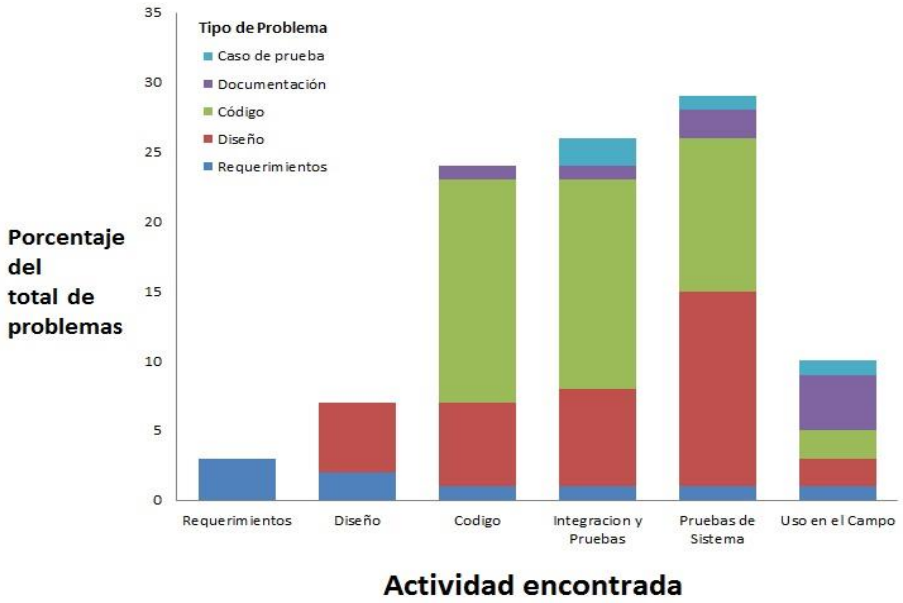

Figura 1. Relación de los Problemas de origen y las fases del Proceso de Desarrollo de Software

En el análisis de Trazabilidad de requerimientos, los modelos existentes tienden a enfocarse en dos aspectos, uno es en la relación entre el código fuente a nivel de programa y el otro es a nivel de proceso tomando en cuenta productos o documentos.

Construcción de escenarios de pruebas de funcionalidad

Formular escenarios de prueba no es sencillo, muchas técnicas se han utilizado en el desarrollo de casos de prueba mediante regresión funcional. Yau y Kishimoto (1987) presentan una técnica basada en el particionamiento de datos de entrada, teniendo como salida grafos causa-efecto. Posteriormente (Chen, 1996) genera un estudio basado en la técnica de particionamiento recursivo para el mantenimiento y entendimiento de programas en lenguaje C++ mediante el paso de mensajes, clases y declaración de dependencias. Un conjunto considerable de las investigaciones se enfocan en técnicas que apoyan la formulación de casos de prueba basados en regresión. En este caso una de las técnicas más utilizadas es la de particionamiento recursivo del código (Huang, 1996) y (Joiner, 1993), otro enfoque es mediante grafos de dependencia (Bates, 1993), (Han, 1997) y el análisis de flujo de control de datos (Agrawal, 1993). El trabajo (Tsai, 2001), se enfoca la trazabilidad como *escenarios basados en regresión funcional*, en este caso el método utilizado es el *fin a fin* (end-to-end E2E), en el cuál la trazabilidad de sistemas de gran escala se proyecta en base de las vistas de los usuarios del sistema. Es un interesante trabajo que se ha convertido en ícono de otros importantes trabajos.

En el trabajo (Kong, 2005) se presentan procesos para estimar la Trazabilidad de requerimientos para sistemas en Web. En este caso la base de estos sistemas es el Framework para Arquitecturas de Aplicaciones Web (*Web Application Architecture Framework*). El modelo que se propone tiene dos aspectos: la perspectiva y su costo. En este caso cada perspectiva se proyecta en términos de estructura, ambiente y localidad. La trazabilidad es modelada entre las abstracciones, la estructura del sistema, el ambiente y la localidad desde el punto de vista de los perfiles de usuario. En cada perspectiva se establecen posibilidades de uso en relación con los patrones de diseño. Aunque el

Framework jamás implementa algún de los patrones que se sugieren en el trabajo, es un estudio interesante que permite establecer la relación de elementos estáticos mediante el análisis de casos de prueba, tomando en cuenta los perfiles operacionales, funcionalidad y el ambiente de operación.

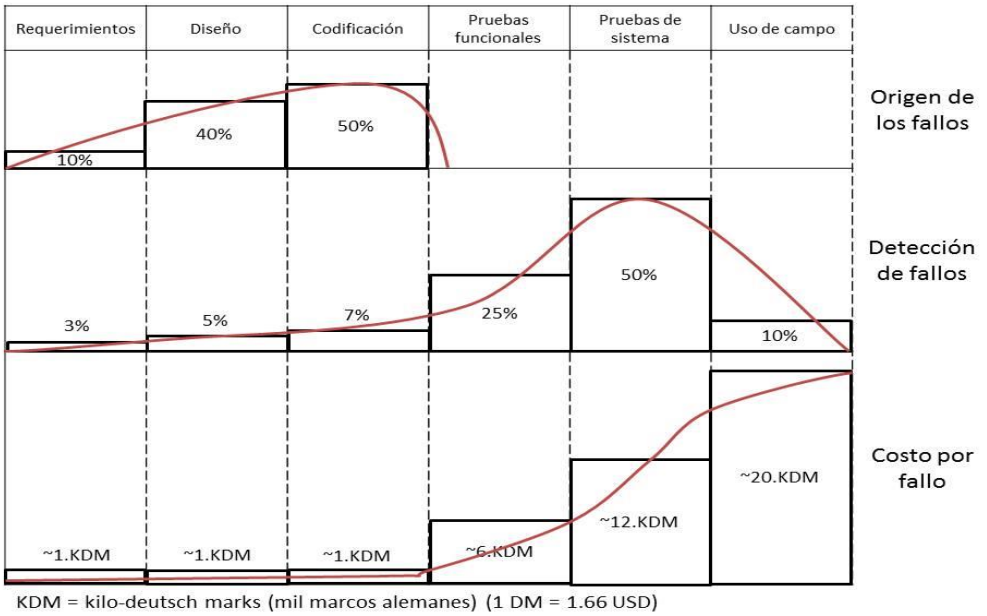

Figura 2. Distribución de origen, detección y coste de los fallos en el proceso de desarrollo.

Uso de patrones de diseño en la construcción de pruebas

Los patrones de diseño como lo describe Erick Gamma (1995), abstraen y resuelven problemáticas que se repiten de forma cotidiana en el diseño de los sistemas de software. Los patrones de diseño han llegado a ser muy populares porque además de su efectividad en el momento de la implementación fomentan el reuso de conceptos de diseño.

En la etapa del diseño de software se da la solución a la especificación de requerimientos, con este enfoque es que el trabajo (Fletcher, 2006), tiene un enfoque interesante para realizar la relación de la funcionalidad de un sistema de software con los patrones de diseño que mejor se adecuan al diseño arquitectónico que soluciona la funcionalidad establecida en la especificación de los requerimientos. Él artículo es sin duda muy interesante porque muestra durante su desarrollo la importancia y la relación de los elementos de funcionalidad con el diseño y su aplicación con los patrones de diseño. Después de analizar estos trabajos es posible observar como algunos autores establecen estudios en los que la propiedad de trazabilidad es modelada mediante los patrones de diseño para representar la funcionalidad de un sistema en particular.

En este trabajo se vislumbra un uso más general de los patrones de diseño; son el medio para generar los casos de prueba de forma dinámica. La importancia de este esquema radica en que su implementación optimiza los recursos de memoria y tiempo que se requieren para construir dichos casos que son centrales en la ejecución de las pruebas. Otro beneficio es que toda la cobertura de pruebas del sistema se fabrica y especializa a tiempo de ejecución.

Herramienta para la Generación Dinámica de Casos de Pruebas Funcionales (Test-Case-Dinamical Tool)

Arquitectura propuesta

En el presente trabajo se propone la arquitectura de una herramienta para la generación dinámica de casos de pruebas de funcionalidad en sistemas de software. La evaluación de los sistemas de software es un proceso costoso, pero en la actualidad a ese costo también hay que incluir que las técnicas tradicionales de evaluación no son suficientes, en particular las tecnologías emergentes complican y extienden el proceso de evaluación de un sistema, por ejemplo en las aplicaciones Web, su naturaleza integra elementos diversos en su infraestructura y arquitectura, lo que complica el proceso de evaluación de funcionalidad.

La presente propuesta se basa en la planeación de pruebas y en la generación de casos de prueba mediante escenarios de regresión funcional. En la Figura 3 se puede apreciar la arquitectura de la *Herramienta para la generación dinámica de casos de pruebas funcionales (Test-Case-Dinamical Tool TCDT por sus siglas en inglés)*. La *TCDT* tiene como base la matriz de trazabilidad de pruebas, que contiene la información de funcionalidad de forma estática, que es la fuente para la generación dinámica de los casos de prueba mediante el uso de los patrones de diseño.

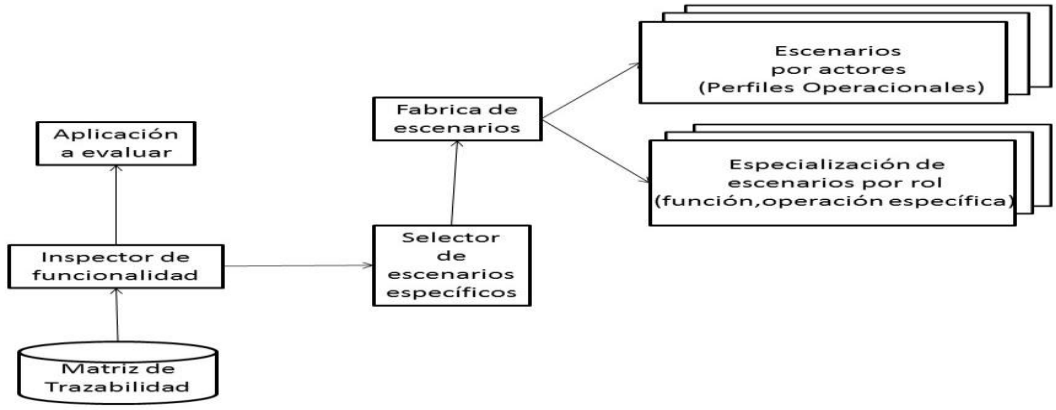

Figura 3: Arquitectura de la Herramienta para la Generación Dinámica de Casos de Prueba funcionales (*Test-Case-Dinamical Tool TCDT*)

En el primer módulo *aplicación a evaluar,* las características de la aplicación de software, así como el propósito, el lenguaje, los sistemas con que se relaciona y el proceso que apoya en el modelo de negocio son los elementos centrales. El *inspector de funcionalidad* tiene implícita la *matriz de trazabilidad*, en ella se encuentran elementos de trazabilidad; perfiles operacionales, los requerimientos funcionales, las operaciones, los componentes y rutas en la arquitectura que resuelven la funcionalidad de estas operaciones. El *selector de escenarios*, toma elementos del caso de prueba que se pretende fabricar como son: el perfil operacional, las operaciones y los componentes en la arquitectura del sistema en relación con la operación, esta información se envía a la fábrica de escenarios como argumentos. La *fábrica de escenarios* toma los argumentos enviados por el selector escenarios y los transforma en productos específicos mediante el patrón de diseño Abstract Factory. El patrón Builder interviene en la especialización del escenario, es decir se toma en cuenta el actor (perfil operacional) y su rol (funcionalidad) para establecer las operaciones que se van a ejecutar.

En la Figura 4 se muestra mediante el diagrama de clases la interacción del sistema con la fábrica abstracta. El selector de escenarios envía los parámetros con los que se generan instancias de los casos de prueba en específico. Para generar cada caso de prueba, el selector envía el perfil operacional y la funcionalidad relacionada a la fábrica. La fábrica genera escenarios (productos) relacionado con los parámetros de funcionalidad que especializa a cada caso de prueba. Este

enfoque automatiza y optimiza los recursos del proceso de evaluación, porque aunque existe una gama muy amplia de casos de prueba sólo se van construyendo los que son ejecutados durante la evaluación.

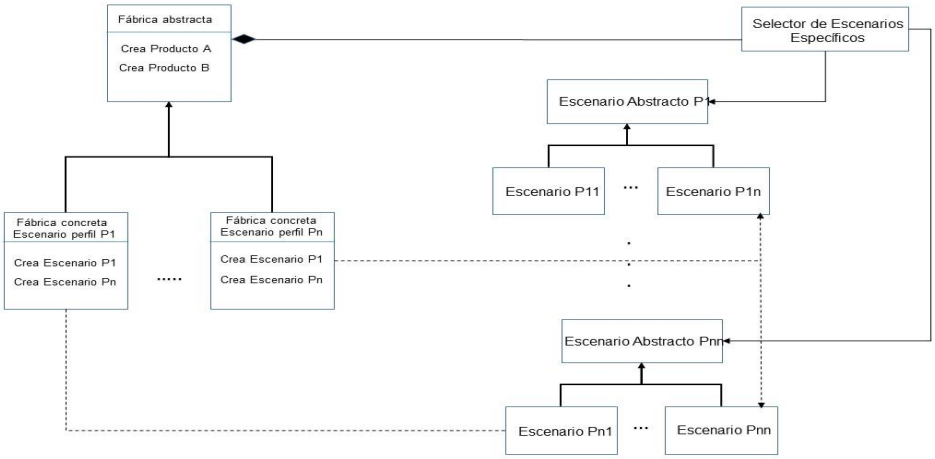

Figura 4. Fabrica abstracta de la Case-Test-Dinamical-Tool

El patrón de diseño Builder hace la especialización en relación a cada una de las operaciones que el sistema debe de cubrir tomando en cuenta los perfiles operacionales asociados. El diagrama de clases se presenta en la Figura 5.

Análisis de resultados.

En el primer prototipo (Dávila-Nicanor, 2005), la elaboración de los casos de prueba era prácticamente manual. De acuerdo al análisis de la trazabilidad, se diseñaban y codificaban todos los casos dentro de la herramienta. En la ejecución de las pruebas, la cobertura estaba limitada por los casos ya programados. Con el actual enfoque, la generación de los casos de prueba tiene nuevos alcances, se generan de forma dinámica de acuerdo a la información de la matriz de trazabilidad de pruebas. Una ventaja muy importante, es que la matriz tiene la posibilidad de actualizase y extenderse. Este enfoque mejora el uso de la herramienta desde cualquier perspectiva. En la evolución de los requerimientos del software, la funcionalidad puede variar de acuerdo al contexto actual de operación del sistema, para generar nuevo casos de prueba sólo es necesario actualizar la matriz de trazabilidad de pruebas (en la Figura 6 se muestra la interfaz principal del sistema para hacer la actualización de la información). El uso de los patrones de diseño Abstract Factory y Builder permiten construir los casos de prueba a tiempo de ejecución de una forma eficiente. Como efecto colateral se optimizaron el tiempo y los recursos de cómputo utilizados en este proceso.

Trabajo a Futuro.

Este trabajo es parte de un *Framework de Evaluación de Confiabilidad* que se ha realizado en Lenguaje Java para sistemas en Internet (Dávila-Nicanor, 2005). El trabajo inmediato es la integración de este estudio con el subsistema *ejecutor de pruebas* para el contexto dinámico. El *ejecutor de pruebas* se ha limitado en versiones anteriores, porque en la formulación de los casos de pruebas fue estática. Con el actual enfoque la construcción es dinámica y se formulan a tiempo de ejecución aquellos casos que realmente se ocupan.

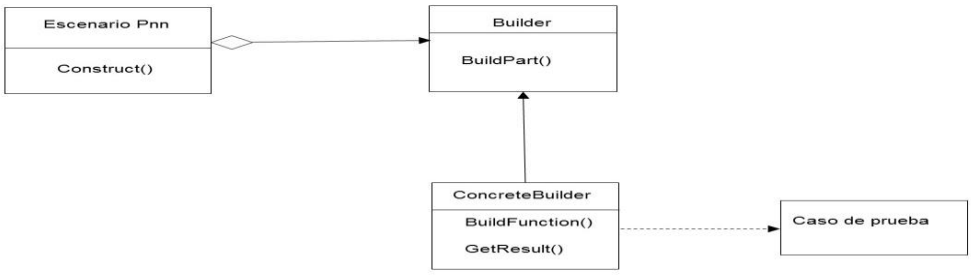

Figura 5. Especialización de Funcionalidad de acuerdo a patrón de diseño Builder.

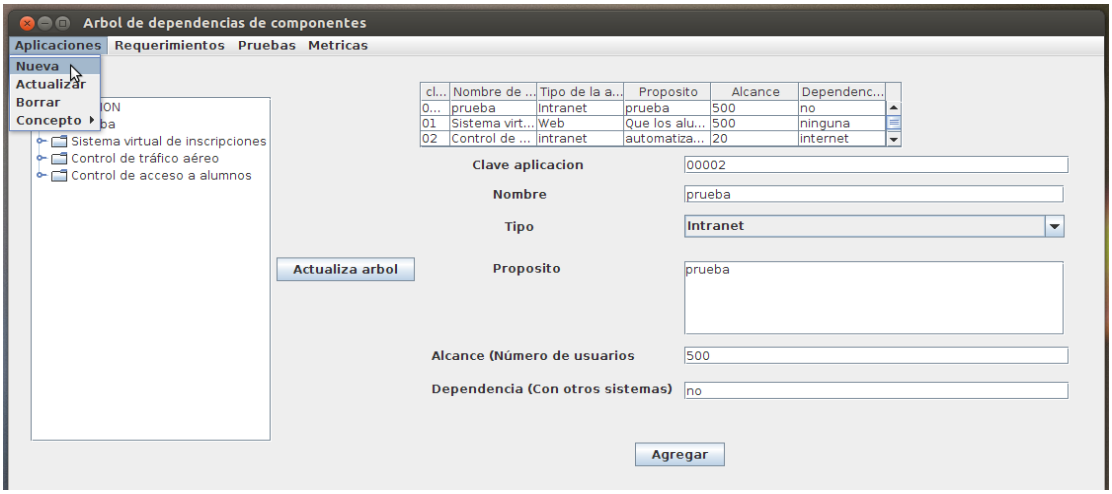

Figura 6. Interfaz de la TCDT para la actualización de información en la matriz de trazabilidad.

Conclusiones.

La evaluación de la funcionalidad de los sistemas de software se basa en la elaboración de casos de prueba mediante escenarios de regresión funcional. Los elementos que intervienen en esta técnica se encuentran en la matriz de trazabilidad de pruebas. Esta matriz es la base que concentra la funcionalidad del sistema, contiene una visión estática del proceso de evaluación. La elaboración de pruebas dinámicas con este enfoque es muy costosa y existen pocos trabajos al respecto. En esta propuesta los elementos de la matriz de trazabilidad de pruebas son la base para generar un conjunto de escenarios dinámicos que son los productos de los patrones de diseño Abstract Factory y Builder. Los patrones mencionados fabrican los escenarios y especializan su funcionalidad en cada caso, tomando en cuenta los perfiles operacionales (actores) y sus funciones (roles). Los resultados del estudio han demostrado que el enfoque es eficiente y confiable en un contexto de ejecución real, optimiza y reduce los recursos empleados en el proceso de pruebas del sistema de software evaluado. Otra ventaja es que si el sistema experimenta cambios en relación con la funcionalidad y se afecta su arquitectura de software, la matriz de trazabilidad puede extenderse y actualizarse desde simples interfaces de usuario, teniendo como resultado la construcción automática de nuevos casos de prueba.

Agradecimientos

Mediante el presente se extienden agradecimientos al Dr. Héctor Rafael Orozco Aguirre, por brindarle su opinión como experto al alumno de licenciatura Omar Marín Guerrero en el diseño del modelo entidad-relación de la base de datos.

Referencias:

1. Agrawal, H. Horgan, J. R. and Krauser, E.W. (1993). Incremental Regresion Testing. Procee-dings of the Conference on Software Maintenance pp. 348-357.
2. Bates, S. and Mansour, N. (1993). Incremental Program Testing Using Program Dependency Graphs. Proc 20th ACM Symp. Of Principies of Programming Languages pp. 384-396.
3. Chen, X. Tsai, W.T. Huang, H. Poonalawa, M. Rayardugan, S. and Wang, Y. (1996). Omega – An Integrated Enviroment for C++ Program Maintenance. Proceedings Int Conf. Software Maintenance pp.114-123
4. Dávila-Nicanor, L. and Mejía-Alvarez, P. (2005). Reliability Evaluation of Web-Based Software Applications. ENC 2005 pp. 106-112
5. Fletcher, J. and Cleland-Huang, J. (2006). Softgoal Traceability Patterns. 17th International Symposium on Software Reliability Engineering (ISSRE'06) IEEE. ISSN:1071-9458 pp. 363-374, Raleigh N.C.
6. Gamma, E. Helm, R. Johnson, R. and Vlissides, J. (2005). Design Patterns. Elements of Reusable Object-Oriented Software. AddisonWesley. Traducción al castellano (2002): Diseño de patrones. Pearson Educación
7. Han, J. (1997). Supporting Impact Analysis and Change Propagation in Software Enginneering Enviroments. In Proceedings of 8th International Workshop on Software Technology and Engineering Practice pp. 32-41.
8. Huang, H. Tsai, W.T. and Subramanian, S. (1996). Generalized Program Slicing for Software Maintenance. Proceedings of International Conference on software and knowledge engineering pp. 261-268.
9. Joiner, J. K. and Tsai, W. T. (1993). Ripple Effect Analysis, Program Slicing and Dependency Analysis. TR-98-84, Computer Science Department, University of Minesota.
10. Kong, X. Liu, L. and Lowe, D. (2005). Web system trace model using a Web application architecture framework, e-Technology, e-Commerce and e-Service. Proceedings. The 2005 IEEE International Conference on, ISBN: 0-7695-2274-2, DOI: 10.1109/EEE.2005.146, IEEE Editorial.
11. Park, R. E. Wolfhart, B. Goethert, William, A. and Florac, B. (1996). Goal-Driven Software Measurement - A Guidebook. Pittsburg, USA: Software Engineering Institute Carnegie Mellon University.
12. IEEE Std. 830, (1988). Sponsor Software Engineering Standards Subcommittee of the Technical Committee on Software Engineering of the IEEE Computer Society , Copyright 1989 by The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017-2394, USA , ISBN 0-7381-0397, SS12542
13. Tsai, W. T. Bai, X. Paul, and R.-Yu, L. (2001). Scenario-Based Functional Regression Testing. Proceeding COMPSAC '01 Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, ISBN:0-7695-1372-7
14. Yau, S.S. and Kishimoto, L. (1987). A Method for Revalidation Modified Program in the Maintenance Phase, Proceedings of IEEE COMPSAC pp. 271-277