Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 9, No. 7, julio, 2016
ISSN: 2306-2495 | RNPS: 2343
Pág. 7-20
http://publicaciones.uci.cu

Tipo de artículo: Artículo original
Temática: Desarrollo de software
Recibido: 1/06/2016 | Aceptado: 01/07/2016

Extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso

Plugin of Visual Paradigm for UML tool for evaluation and correction of Use Case Diagram

Lionel R. Baquero Hernández [1*], Dayana Mendoza Peña [1], Osviel Rodríguez Valdés [1], Omar Mar Cornelio [1]
[1] Facultad 6. Universidad de las Ciencias Informáticas, Carretera a San Antonio de los Baños, km2 [1/2], Torrens, Boyeros, La Habana, Cuba. CP.: 19370
[*] Autor para correspondencia: lrbaquero@estudiantes.uci.cu

Resumen

En el presente trabajo se propone una extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso. Para su desarrollo su utilizó la metodología OpenUP, el Lenguaje de Modelado Unificado en su versión 2.5, Visual Paradigm for UML 8.0 como herramienta de modelado, el lenguaje de programación Java 8.0 y el Entorno de Desarrollo Integrado NetBeans 8.0. Además, se utilizó la biblioteca OpenAPI para poder extender las funcionalidades del Visual Paradigm. A la aplicación se le aplicaron pruebas funcionales a nivel de desarrollador, mediante el método de Caja Negra y con la técnica de Partición de Equivalencia, resolviendo las no conformidades detectadas satisfactoriamente. Como resultado de la investigación se obtuvo una extensión capaz de detectar y corregir errores conceptuales cometidos en el modelado de Diagramas de Casos de Uso.

Palabras clave: Diagramas de Casos de Uso; evaluación de Diagramas de Casos de Uso; corrección de Diagramas de Casos de Uso, Visual Paradigm, Lenguaje de Modelado Unificado.

Abstract

In this work is proposed an extension to Visual Paradigm for UML tool for evaluation and correction of Use Case Diagram. It is expected to contribute to the elimination of errors common concepts in modeling these types of diagrams. It was used OpenUP as methodology for the development, the Unified Modeling Language in its version

2.5, the Visual Paradigm for UML 8.0 as modeling tool, the programming language Java 8.0 and NetBeans Integrated Development Environment 8.0. In addition, the Open API library used to extend the functionality of Visual Paradigm. The application functional tests were applied at the level of developer by the method of Black Box and the Equivalence partitioning technique, solving successfully detected nonconformities. As a result of the investigation was obtained an extension able to detect and correct errors in modeling Use Case Diagram.

Keywords: *Use Case Diagram; evaluation of Use Case Diagram; correction of Use Case Diagram; Visual Paradigm; Unified Modeling Language*

Introducción

En el proceso de Análisis del Software de la disciplina Ingeniería de Software se realiza la Ingeniería de Requisitos que es el uso sistemático de procedimientos, técnicas, lenguajes y herramientas para obtener el análisis, documentación, evolución continua de las necesidades del usuario y la especificación del comportamiento externo de un sistema que satisfaga las necesidades del usuario. Una vez recopilados los requisitos se crean un conjunto de escenarios que identifiquen una línea de utilización para el sistema que va a ser construido, estos escenarios son llamados casos de uso y facilitan la descripción de cómo el sistema se usará. En general, un caso de uso es, simplemente, un texto escrito que describe el papel de un actor que interactúa con el acontecer del sistema (PRESSMAN, 2002).

Los Diagramas de Casos de Uso (DCU) son una técnica para capturar requisitos o información de cómo un sistema o negocio trabaja, y están compuesto por los casos de uso, los actores que se pueden definir como algo con comportamiento, como una persona (identificada por un rol), sistema informatizado u organización (LARMAN, 2003) y las relaciones existentes entre ellos.

Actualmente existen una gran variedad de herramientas CASE (Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadora) para el proceso de desarrollo de software. Las herramientas CASE, están tomando cada vez más relevancia en la planeación y ejecución de proyectos que involucren sistemas de información, pues suelen inducir a sus usuarios a la correcta utilización de metodologías que le ayudan a llegar con facilidad a los productos de software construidos. Todas las herramientas CASE prestan soporte a un lenguaje de modelado para acompañar la metodología y es lógico suponer, que un alto porcentaje de ellas soportan el Lenguaje de Modelado Unificado (UML). Esto se debe a la amplia aceptación de este lenguaje, su valor conceptual y visual, así como su facilidad para ser extendido para representar elementos particulares a determinados tipos de aplicaciones

(QUINTERO et al., 2012).

En la Universidad de las Ciencias Informáticas (UCI) se ha estandarizado el uso del Visual Paradigm for UML en su distribución libre como herramienta CASE para el modelado de los procesos de desarrollo de software que en ella se llevan a cabo, dado por la gran cantidad de ventajas que posee, las cuales están en concordancia con los intereses y políticas establecidas en la institución. Entre sus principales características se encuentran que es multiplataforma, posee interoperabilidad, facilita la colaboración en equipo y brinda apoyo al ciclo de vida completo del desarrollo de software (ROSALES et al., 2013).

Entre los diversos tipos de diagramas que el Visual Paradigm for UML permite modelar se encuentran los DCU. En el modelado de estos tipos de diagramas se pueden cometer numerosos errores de concepto que pueden ser contraproducentes, generando pérdidas de tiempo, así como retrasos en los cronogramas de proyecto. Estos errores pueden traducirse, en su caso más crítico, en no conformidades en la fase de pruebas al sistema. Esta herramienta no permite hacer una evaluación y corrección de los mismos, para servir como referente al usuario que está diseñando el software en cuestión. Para darle solución a este problema se pretende desarrollar una extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso. Se espera que contribuya a la eliminación de los errores de conceptos frecuentes en el modelado de estos tipos de diagramas.

Materiales y métodos

Como fundamentos principales de la presente investigación, se tuvieron en cuenta un conjunto de definiciones, asociadas al Lenguaje de Modelado Unificado, los Diagramas de Casos de Uso y la evaluación de estos últimos. Se seleccionaron las herramientas y las tecnologías que más se ajustaban a las necesidades reales del desarrollo de la extensión. Se tuvieron en cuenta además las características del equipo de desarrollo y la complejidad de la aplicación para la selección de una metodología acorde a estas características.

Lenguaje de Modelado Unificado

El Lenguaje Unificado de Modelado (UML, Unified Modeling Language) fue adoptado en noviembre de 1997 por OMG (Object Management Group) como una de sus especificaciones y desde entonces se ha convertido en un estándar de facto para visualizar, especificar y documentar los modelos que se crean durante la aplicación de un proceso de software. UML ha ejercido un gran impacto en la comunidad del software, tanto a nivel de desarrollo como de investigación (GARCÍA et al., 2004).

UML es empleado para la documentación de proyectos, actualmente se encuentra en la versión 2.5. No dispone de una metodología de desarrollo, por lo cual se apoya en metodologías de terceros para el desarrollo de proyectos. Según (OMG, 2015) UML modela diagramas de actividad, clases, comunicación, componentes, estructura compuesta, despliegue, interacción, objetos, paquetes, perfil, máquina de estado, tiempo y casos de uso.

Diagramas de Casos de Uso

Los principales elementos que componen un Diagrama de Casos de Uso son: actores, casos de uso y las relaciones existentes entre ellos. Para el tratamiento de los mismos se tomarán las definiciones ofrecidas por (RUMBAUGH, 2007).

Un Diagrama de Casos de Uso (DCU) muestra las relaciones existentes entre actores y casos de uso dentro de un sistema. El actor es una abstracción de las entidades externas a un sistema, subsistemas o clases que interactúan directamente con el sistema. Un actor participa en un caso de uso o conjunto coherente de casos de uso para llevar a cabo un propósito global. Es una idealización con un propósito y significado concretos, que puede no corresponderse con objetos físicos. Un objeto físico puede combinar propósitos dispares y por tanto ser modelado por varios actores y viceversa, diferentes objetos físicos podrían incluir el mismo propósito, y por tanto ser modelados como el mismo actor.

Las diferentes interacciones de los actores con un sistema se cuantifican en casos de uso. Un caso de uso es una especificación de las secuencias de acciones, incluyendo secuencias variantes y secuencias de error, que pueden ser efectuadas por un sistema, subsistema o clase por interacción con actores externos. Describe una secuencia completa que es iniciada por un actor. Entre los elementos que componen un DCU pueden existir relaciones que son una conexión semántica materializada. Entre las clases de relaciones presentes en un DCU se cuentan la asociación o comunicación, inclusión, extensión y generalización o especialización.

Evaluación de Diagramas de Casos de Uso

La evaluación de diagramas es un proceso diseñado específicamente para validar el modelado de estos. Debe estar encaminada a determinar si los elementos del diagrama están correctamente representados y relacionados entre sí. Este proceso de evaluación debe estar estructurado de forma tal que sea capaz de detectar los errores cometidos en el modelado del diagrama. Como salida de este proceso se debe obtener la lista de errores cometidos en el modelado del diagrama, la cual será la entrada del proceso de corrección.

A partir de las definiciones estudiadas sobre los elementos asociados a un DCU y de encuestas realizadas a expertos

(especialistas y profesores) del área de la Ingeniería de Software, se pudieron identificar los errores que pueden ser cometidos en el modelado de estos diagramas. Luego del estudio riguroso de los resultados de la investigación realizada, se concluyó que se pueden asumir como errores en el modelado de DCU:

- Ausencia de algún elemento del diagrama.
- Elemento sin relacionar con otro en el diagrama.
- Elemento incorrectamente relacionado con otro en el diagrama.
- Existencia de relaciones invertidas en el diagrama.
- Elementos que comparten relaciones múltiples.

En un DCU deberá estar presente al menos una instancia de cada uno de sus elementos (Actor, Caso de Uso y Relación), dado que con la ausencia de uno de estos el diagrama pierde sentido y pertinencia. Puede suceder que exista algún elemento del diagrama que no se relacione con ningún otro, en este caso se ha cometido un error en el modelado del DCU.

Que un Actor o Caso de Uso esté incorrectamente relacionado con otro elemento, es la utilización incorrecta de las posibles relaciones entre estos elementos. Los Actores solo se relacionarán con otros Actores a través de la relación de tipo generalización o especialización. Un Actor se relacionará con un Caso de Uso mediante una comunicación. La relación entre Casos de Uso será posible con la utilización de las relaciones inclusión y extensión.

Si las relaciones han sido correctamente utilizadas, pero la dirección de estas no es la definida para relacionar los elementos, se puede afirmar que dicha relación está invertida. Además, es válido aclarar que, entre dos elementos del diagrama, solo podrá existir una relación, evitando de esta forma las relaciones múltiples.

La evaluación de un DCU deberá realizarse de forma tal que el fin de este proceso sea validar la existencia o no de los errores en el modelado del diagrama. La entrada del proceso de evaluación debe ser el DCU y la salida de este debe ser la lista de errores identificados y de ser posible las sugerencias necesarias para la corrección de los mismos.

Metodología de desarrollo de software

El desarrollo de la extensión de la herramienta Visual Paradigm for UML para la evaluación y corrección de Diagramas de Casos de Uso se apoya en un enfoque Ágil. De ahí se decide emplear la metodología de desarrollo OpenUP (Open Unified Process) dado que es una forma de desarrollo más ágil y ligera (SAAVEDRA et al., 2013). Además, que permite colaborar para sincronizar intereses y compartir conocimiento, equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto, centrarse en la arquitectura de forma temprana

para minimizar el riesgo y organizar el desarrollo, y un desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo (BALDUINO, 2007).

Todo proyecto en OpenUP consta de cuatro fases: inicio, elaboración, construcción y transición. Cada una de estas fases se divide a su vez en iteraciones, lo cual tiene como ventaja que permite a los integrantes del equipo de desarrollo aportar con micro-incrementos, que pueden ser el resultado del trabajo de unas pocas horas o unos pocos días. Además, este ciclo de vida provee a los clientes de una visión del proyecto, transparencia y los medios para que controlen la financiación, el riesgo, el ámbito y el valor de retorno esperado (HERNÁNDEZ & RODRIGUEZ, 2015). La selección de la metodología de desarrollo OpenUP, estuvo fundamentada en que es muy apropiada para proyectos pequeños y de bajos recursos, como es el caso del presente equipo de desarrollo, permitiendo de esta forma disminuir las probabilidades de fracaso e incrementar las probabilidades de éxito, mediante la detección de errores tempranos a partir de su ciclo iterativo, lo cual supone una ventaja importante en el desarrollo de la propuesta de solución.

Lenguaje de modelado para el desarrollo de la extensión

Como lenguaje de modelado se seleccionó UML en su versión 2.5, dado que es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es una consolidación de muchas de las notaciones y conceptos más usados en la metodología orientada a objetos. Ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios, funciones del sistema, aspectos concretos como expresiones de lenguajes de programación, componentes de software reutilizables y esquemas de bases de datos. Además, permite especificar y visualizar un sistema. Se puede utilizar para definir un sistema de software, detallar los artefactos, especificar su documentación y construcción. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software, contando con varios tipos de diagramas para llegar a representar los diferentes puntos de vistas de un sistema (TRUJILLO et al., 2013).

Herramienta CASE

Para el modelado se utilizará Visual Paradigm for UML 8.0 por ser una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. Agiliza la construcción de aplicaciones con calidad y a un menor coste. Posibilita la generación de bases de datos, transformación de diagramas de Entidad-Relación en tablas de base de datos, así como ingeniería inversa de bases de datos (RONDÓN et al., 2011).

Lenguaje de programación

Entre los lenguajes de Programación Orientada a Objetos (POO) que existen en la actualidad se encuentran C, C++, C#, PHP y otros donde el más utilizado es Java. Java desarrollado por Sun Microsystems, ha evolucionado rápidamente en el ámbito de las aplicaciones a gran escala. Es un lenguaje de programación de alto nivel, del cual uno de sus pilares es la posibilidad de ejecutar un mismo programa en diversos sistemas operativos (independiente de la plataforma). Su diseño presenta como características ser robusto, seguro, portable, independiente a la arquitectura, dinámico e interpretado (DEITEL, 2004). En la implementación de la extensión es utilizado Java 8.0 por ser el lenguaje que permite tener acceso a las funcionalidades administradas por la Interfaz de Programación de Aplicaciones (OpenAPI).

Entorno de Desarrollo Integrado

En la implementación de la propuesta de solución se define utilizar el IDE (Integrated Development Enviorment) NetBeans 8.0 que es una herramienta de código abierto con una gran base de usuarios, una comunidad en constante crecimiento. Existe además un número importante de módulos para extenderlo. Además, es un producto libre y gratuito sin restricciones de uso. Está compuesta por una base modular y extensible usada como una estructura de integración para crear grandes aplicaciones de escritorio.

Entre sus características están las administraciones de ventanas, almacenamiento, interfaces y configuraciones de usuario. Soporta el desarrollo de aplicación Java (J2SE, web, EJB y aplicaciones móviles), empresariales con Java EE 5, JavaFX 2.0, WebLogic 12c y CSS3. Empresas independientes asociadas, especializadas en desarrollo de software, proporcionan extensiones adicionales que se integran fácilmente en la plataforma y que pueden también utilizarse para desarrollar sus propias herramientas y soluciones (FUNES, 2008).

Interfaz de Programación de Aplicaciones

Para el desarrollo de la extensión es necesaria la utilización de una Interfaz de Programación de Aplicaciones (API – Application Programming Interface), que es una interfaz de comunicación entre componentes de software. Consiste en proporcionar un conjunto de funciones de uso general, que son llamadas a bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación. En este caso se utilizará OpenAPI que es un mecanismo de extensión proporcionada por Visual Paradigm para extender las funcionalidades del software. El API es basado en Java y permite tener acceso completo a los datos del modelo en el archivo de proyecto. Mediante el uso de este mecanismo se pueden implementar algunas funciones personalizadas

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 9, No. 7, julio, 2016
ISSN: 2306-2495 | RNPS: 2343
Pág. 7-20
http://publicaciones.uci.cu

llamadas plugins o extensiones para lograr determinados fines sobre la herramienta (TRUJILLO et al., 2013).

Resultados y discusión

Como resultado del proceso de desarrollo del software se obtuvo una extensión de la herramienta Visual Paradigm for UML. Con esta extensión se pueden evaluar y corregir los DCU, de forma tal que con ella se reducen los errores de concepto frecuentemente reflejados en su modelado. La extensión se desarrolló teniendo en cuenta tres fases para la evaluación y corrección de un DCU: obtención de los datos del DCU, evaluación del DCU y corrección del DCU (ver Figura 1).

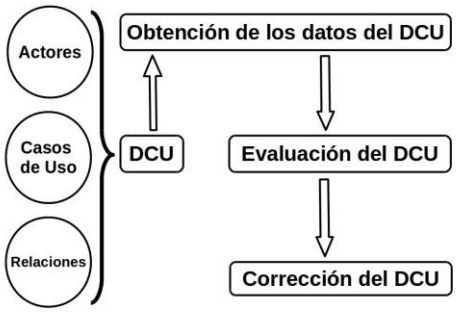

Figura 1. Fases de la evaluación y corrección de un DCU.

Obtención de los datos

La obtención de los datos comienza cuando el usuario accede a la opción Evaluar DCU a través del clic derecho sobre el contexto del diagrama o a través del menú: Herramientas/Evaluar DCU. Para esto se construye una instancia de la clase controladora ActionController, la cual se encarga de construir la interfaz visual UserInterface, pasándolo por parámetro una instancia de la clase MetodoEvaluacionController, encargada de evaluar los DCU modelados. A esta interfaz también se le pasa por parámetro una instancia de la clase MetodoCorreccionController, encargada de hacer las correcciones a los errores detectados.

Todo esto es posible con la utilización de la clase Action, al implementar los métodos *performAction* y *update* de la clase VPContextActionController ofrecida por el API para actualizar las acciones que se realizan en la herramienta. La creación de la interfaz incluye mostrar el listado de los DCU modelados con la herramienta (ver Figura 2).

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
seriecientifica@uci.cu

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 9, No. 7, julio, 2016
ISSN: 2306-2495 | RNPS: 2343
Pág. 7-20
http://publicaciones.uci.cu

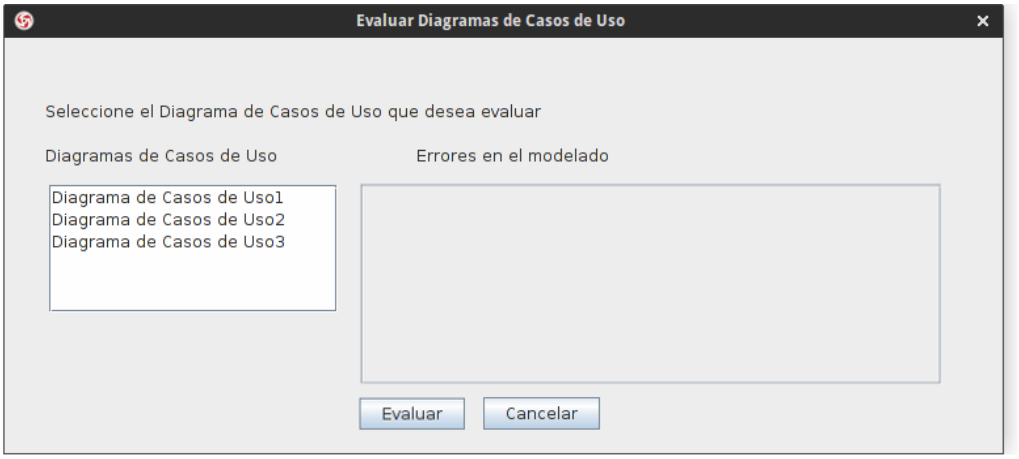

Figura 2. Interfaz de usuario de la extensión.

Para el proceso de obtención de los datos de un DCU modelado, es necesario el uso de un conjunto de componentes ofrecidos por el OpenAPI, como se describe en la Tabla 1.

Tabla 1. Descripción de los componentes utilizados para la obtención de los datos.

Componente	**Descripción**
DiagramManager	Interfaz que permite el control de los diagramas modelados.
IDiagramUIModel	Interfaz que define los diagramas modelados.
IUseCaseDiagramUIModel	Interfaz que define los DCU modelados.
IModelElement	Interfaz que define un elemento del DCU.
IDiagramElement	Interfaz que define los elementos de un diagrama.
IDiagramProperty	Interfaz que define las propiedades de un diagrama.

Evaluación del DCU

La Evaluación de un DCU inicia cuando el usuario una vez mostrada la interfaz, selecciona un diagrama y la opción Evaluar. La finalidad del proceso de evaluación es obtener el listado de los errores en el modelado del DCU, para que estos puedan ser mostrados al usuario. Una vez obtenido el listado de errores, se cargan en la interfaz UserInterface. Por cada error mostrado en pantalla se da una sugerencia para que si el usuario lo desea pueda corregirlo manualmente (ver Figura 3).

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 9, No. 7, julio, 2016
ISSN: 2306-2495 | RNPS: 2343
Pág. 7-20
http://publicaciones.uci.cu

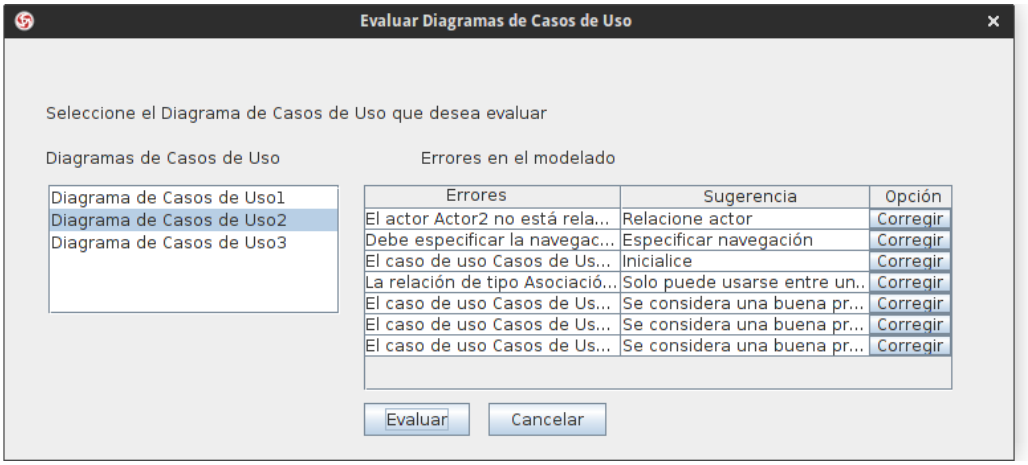

Figura 3. Interfaz de usuario de la extensión con los resultados de la evaluación de un DCU.

En la evaluación es utilizada la clase MetodoEvaluacionController, la cual obtiene la lista de los errores cometidos en el modelado del DCU, a través del método *listaErrores*. Este método crea una instancia de la clase MetodoEvaluacion, en la que se hacen todas las validaciones necesarias para la evaluación del diagrama. En la Tabla 2 se muestran los principales métodos de la clase MetodoEvaluacion.

Tabla 2. Descripción de los métodos de la clase MetodoEvaluacion para la evaluación del DCU.

Métodos	**Parámetros**	**Descripción**
comprobaciones	LinkedList<IModelElement> actores, LinkedList<IModelElement> casosUso, LinkedList<IModelElement> elementos, LinkedList<IModelElement> relaciones	Se encarga de hacer llamadas a los otros métodos de la clase y devuelve el listado de los errores cometidos en el modeldo.
identificarDiagramaSinActor	LinkedList<IModelElement> actores	Valida si existe al menos un actor en el diagrama, en caso de que no exista retorna un error.
identificarDiagramaSinCasoUso	LinkedList<IModelElement> casosUso	Verifica la existencia de al menos un caso de uso en el diagrama, en caso de que no exista retorna un error.
identificarElementosSinRelacionar	LinkedList<IModelElement> elementos	Evalúa si todos los elementos han sido relacionados con al menos un elemento del diagrama, en caso contrario retorna un error.
identificarRelacionesIncorrectas	LinkedList<IModelElement> elementos	Es responsable de evaluar que las relaciones entre los elementos del

		diagrama han sido correctamente modeladas, y la dirección de las mismas. En caso de que exista alguna relación incorrectamente modelada se retorna un error
identificarRelacionesMultiples	LinkedList<IModelElement> actores, LinkedList<IModelElement> casosUso,	Verifica que no existan relaciones múltiples entre dos elementos del diagrama, en caso afirmativo retorna un error.

Corrección del DCU

Las correcciones del DCU deben hacerse una para cada uno de los errores listados en la interfaz. Este proceso comienza cuando el usuario selecciona la opción Corregir de un error mostrado en la interfaz. Al seleccionarlo se muestra otra ventana que muestra las posibles opciones y las sugerencias (ver Figura 4).

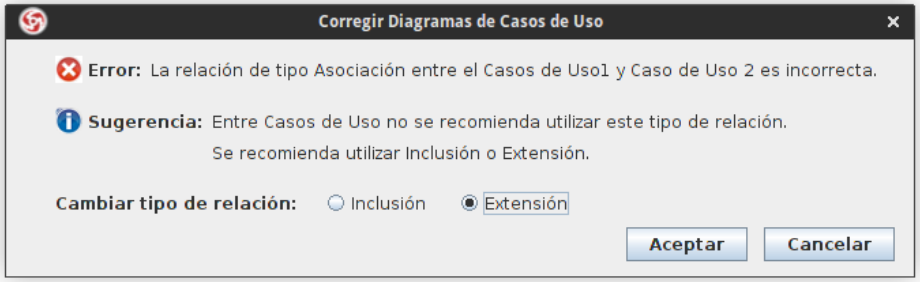

Figura 4. Interfaz de usuario de la extensión para la corrección de un error detectado.

En la Corrección es utilizada la clase MetodoCorreccionController, en la que se implementan los métodos necesarios para corregir cada uno de los errores detectados. En esta clase se implementa un método complementario a cada uno de los métodos de la clase MetodoEvaluacion. Además, contiene algunos métodos claves para la modificación de los diagramas, responsables de crear, modificar y eliminar elementos de este.

Pruebas del Software

La evaluación del sistema se realizó durante todo el ciclo de desarrollo de la extensión, mediante el nivel de prueba: Pruebas de Desarrollador (DUSTIN et al., 1999). Estas son diseñadas e implementadas por el equipo de desarrollo. Todo esto utilizando el método de prueba de Caja Negra (MYERS et al., 2011). La técnica de prueba seleccionada en este caso es la Partición de Equivalencia (BURNSTEIN, 2006), que tiene como principal objetivo la definición de casos de pruebas que descubran clases de errores, reduciendo así el número de clases de pruebas a desarrollar.

Serie Científica de la Universidad de las Ciencias Informáticas
Vol. 9, No. 7, julio, 2016
ISSN: 2306-2495 | RNPS: 2343
Pág. 7-20
http://publicaciones.uci.cu

Este método es conocido también como Pruebas de Comportamiento y se ejecuta por cada caso de uso usando datos válidos e inválidos, para verificar que los resultados esperados se correspondan con los mensajes apropiados de error o precaución respectivamente. Las pruebas se realizan sobre la interfaz de usuario del software. Los casos de prueba de Caja Negra constituyen un conjunto de condiciones de ejecución desarrollado para cumplir un objetivo en particular o una función esperada y demostrar el estado de operatividad en que se encuentra el software. En el gráfico de la Figura 5 se muestran los resultados de la aplicación de las pruebas funcionales al sistema.

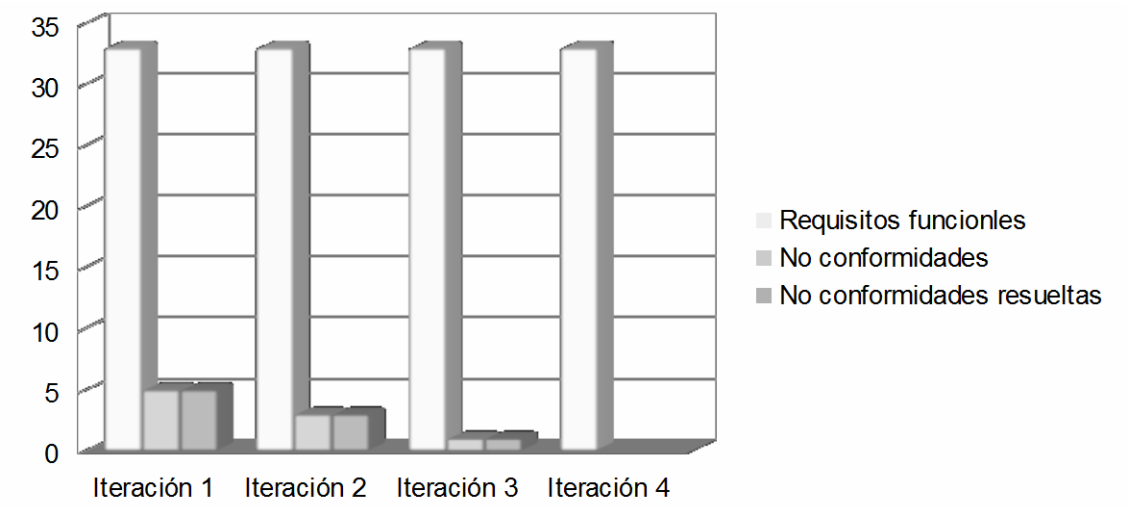

Figura 5. Resultados de las iteraciones de las pruebas funcionales.

Las pruebas se desarrollaron en tres iteraciones, teniendo treinta y tres requisitos funcionales. En la primera iteración se obtuvieron 5 no conformidades, que fueron resueltas inmediatamente. En la segunda iteración se detectaron 3 nuevas no conformidades, para la cuales se encontró solución. Lo mismo sucedió en la tercera iteración donde se encontró una no conformidad. Para la cuarta iteración no se detectaron nuevas no conformidades, por lo que se decidió no seguir realizando iteraciones.

Conclusiones

A partir de la bibliografía consultada y de las principales definiciones de evaluación, se pudo elaborar una definición propia de la evaluación de diagramas para su utilización en la investigación. El análisis de la versión 2.5 de UML y las encuestas realizadas permitieron definir los errores conceptuales que pueden ser cometidos en el modelado de un DCU. Con el estudio de las herramientas y tecnologías modernas en el desarrollo de aplicaciones de este tipo, se

Editorial "Ediciones Futuro"
Universidad de las Ciencias Informáticas. La Habana, Cuba
seriecientifica@uci.cu

pudieron recopilar los basamentos teóricos suficientes para la selección de aquellas que se ajustan más a las necesidades reales de la propuesta de solución.

Para el desarrollo de la extensión se propuso como solución un diseño de tres fases o procesos: obtención de los datos, evaluación y corrección, en las que se reúne el modo de implementación y su funcionamiento. A la aplicación se le aplicaron pruebas funcionales a nivel de desarrollador, mediante el método de Caja Negra y con la técnica de Partición de Equivalencia. Las no conformidades detectadas durante este proceso fueron resultas satisfactoriamente. Esta extensión es capaz de detectar y corregir cada uno de los errores conceptuales frecuentemente cometidos en el modelado de un DCU. Con esta aplicación se puede evitar pérdidas de tiempo, retrasos en los cronogramas de proyecto y no conformidades en el desarrollo de un software.

Referencias

- BALDUINO, R. Introduction to openup. 2007. [En línea] Eclipse, 2015 [Consultado el: 27 de octubre de 2015]. Disponible en: [http://www.eclipse.org/epf/general/Op enUP.pdf].
- BURNSTEIN, I. Practical software testing: a process-oriented approach. Berlin, Springer, 2006. 710p.
- DEITEL, H. M. & DEITEL, P. J. Cómo programar en Java. Madrid, Pearson Educación, 2004. 1227p.
- DUSTIN, E. et al. Automated software testing: introduction, management, and performance. Boston, Addison-Wesley Professional, 1999. 608p.
- FUNES, L. Conociendo a NetBeans Platform: Introducción. 2008. [En línea] Le Funes, 2015 [Consultado el: 15 de abril de 2016]. Disponible en: [http://wiki.netbeans.org/ConociendoNetbeansPlatformIntroduccion].
- GARCÍA, J. J. et al. UML: el lenguaje estándar para el modelado de software. Novática: Revista de la Asociación de Técnicos de Informática, 2004, no 168, p. 4-5.
- HERNÁNDEZ, B. & RODRIGUEZ, O. Sistema para la gestión del proceso de impresiones del Vicedecanato de Administración de la facultad 6 de la Universidad de las Ciencias Informáticas. Tesis para optar por el Título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana, 2015.
- LARMAN, C. UML y Patrones. Madrid, Pearson Educación, 2003. 624p.
- MYERS, G. J. Et al. The art of software testing. New Jersey, John Wiley & Sons, 2011. 256p.
- OMG. Unified Modeling Lenguage, Versión 2.5. Massachusetts, OMG, 2015. 794p.
- PRESSMAN, R. S. Ingeniería de Software, Un Enfoque Práctico. New York, McGraw-Hill Companies, 2002. 640p.
- QUINTERO, J. B. et al. Un estudio comparativo de herramientas para el modelado con UML. Revista Universidad

EAFIT, 2012, 41(137), p. 60-76.

- RONDÓN, Y. et al. Diseño de la base de datos para sistemas de digitalización y gestión de medias. Revista de Informática Educativa y Medios Audiovisuales, 2011, 8(15), p. 17-25.

- ROSALES, Y. et al. Extensión de la herramienta Visual Paradigm para la generación de clases de acceso a datos con Doctrine 2.0. Serie Científica, 2013, 6(10), p. 1-16.

- RUMBAUGH, J. et al. El Lenguaje Unificado de Modelado. Manual de Referencia. Denmark, Addison-Wesley, 2007. 688p.

- SAAVEDRA, D. et al. Aplicación web para la realización de estudios farmacocinéticos, versión 2.0. Revista Cubana de Informática Médica, 2013, 5(2), p. 118-131.

- TRUJILLO, A. et al. Extensión de la herramienta Visual Paradigm para la generación de las clases de acceso a datos con Doctrine 2.0. Tesis para optar por el Título de Ingeniero en Ciencias Informáticas, Universidad de las Ciencias Informáticas, La Habana, 2013.