Revista
Española de

Innovación, Calidad e Ingeniería del Software

REICIS

Volumen 3, No. 3, diciembre, 2007

Web de la editorial: www.ati.es
E-mail: reicis@ati.es
ISSN: 1885-4486

Publicado por la Asociación de Técnicos en Informática

Revista Española de Innovación, Calidad e Ingeniería del Software (REICIS)

Contenidos

REICIS

Editorial

REICIS

Este número de diciembre de 2007 de REICIS consolida la sección de contribuciones invitadas aportadas por expertos de la industria y del mercado ya iniciada en el número anterior que nos transmitirán, en breves reflexiones, las tendencias o los temas de actualidad que perciban como más interesantes desde su posición privilegiada en el ámbito de la ingeniería y la calidad del software. En esta ocasión, contamos con Pablo Pérez San José que es gerente del Observatorio de la Seguridad de la Información en el Instituto Nacional de Tecnologías de la Comunicación (INTECO). En su contribución nos informa de las iniciativas que esta entidad está promoviendo en el ámbito de la calidad del software en España.

Por otra parte, la sección regular de artículos se nutre de dos contribuciones que son fruto del acuerdo con la organización del Taller sobre Pruebas en Ingeniería del Software PRIS 07 celebrado en Zaragoza el 11 de septiembre de 2007 en el marco de las Jornadas de Ingeniería del Software y Bases de Datos y del macrocongreso CEDI. El proceso incluye la selección de los originales más prometedores para posteriormente solicitar a sus autores una versión mejorada y extendida. Esta versión es finalmente revisada por el comité editorial solicitando las mejoras necesarias para su publicación en REICIS. Para completar la cobertura de esta reunión científica, este número incluye una reseña sobre el taller PRIS'07 a cargo de su responsable, Javier Tuya, de la Universidad de Oviedo.

Por último, queremos aprovechar este editorial para informar de la próxima incorporación de REICIS a **Redalyc** (http://redalyc.uaemex.mx/), la Red de Revistas Científicas de América Latina y el Caribe, España y Portugal. Esta hemeroteca digital supone la referencia básica sobre revistas y publicaciones científicas y técnicas de calidad en el ámbito español e latinoamericano. De esta manera, REICIS expande su difusión tras su adscripción a la iniciativa e-revistas del Centro en la primavera de 2007 pertenencia al portal **E-revistas** (http://www.erevistas.csic.es) auspiciado por dos centros del CSIC (Consejo Superior de Investigaciones Científicas), el Centro de Información y Documentación Científica (CINDOC: http://www.cindoc.csic.es/) y Centro Técnico de Informática (CTI: http://www.cti.csic.es/). Este portal recoge a las publicaciones científicas y técnicas de España publicadas en formato de acceso abierto y que han superado la verificación de los criterios de calidad establecidos por sus gestores. Aprovechameos esta ocasión para hacer un llamamiento especial a autores iberoamericanos que trabajen en las distintas áreas de la innovación, la calidad y la ingeniería del software para que remitan sus contribuciones a REICIS.

En este sentido, el conjunto de revistas del portal E-revistas ha remitido una carta a la Comisión Nacional de Evaluación de la Actividad Investigadora (CNEAI) del Ministerio de Educación y Ciencia para que incluya explícitamente a las publicaciones del portal en el catálogo de revistas de calidad en las que la publicación de trabajos se considera un mérito

investigador: la razón de esta petición es que los criterios fijados por la inclusión en el portal son idénticos a los fijados por esta comisión para considerar a una revista como publicación de calidad. Creemos que debe reconocerse el trabajo realizado por estas publicaciones que, además, proporcionan acceso libre y sin restricciones al conocimiento frente al lucro comercial presente en otros catálogos como el de ISI (http://isiknowledge.com/), de la empresa Thomson, que es el explícitamente mencionado en los criterios de evaluación.

Luis Fernández Sanz
Juan J. Cuadrado-Gallego
Editores

Presentación

REICIS

Este segundo número de REICIS del año 2007 publica, tras el proceso de revisión de nuestro comité editorial, dos contribuciones extendidas y revisadas seleccionadas de entre las remitidas al taller PRIS'07.

En este caso, el primero de los trabajos publicados corresponde al titulado "Generación e implementación de pruebas del sistema a partir de casos de uso" cuyos autores son Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Arturo H. Torres y Jesús Torres de la Universidad de Sevilla. En este artículo se presenta un método para la generación de pruebas a partir de casos de uso basado en la referencia del perfil de pruebas de UML.

El segundo trabajo se titula "Estrategia de gestión de las pruebas funcionales en el Centro de Ensayos de Software" y ha sido elaborado por Beatriz Pérez Lamancha del Centro de Ensayo de Software de Uruguay. Describe una estrategia aplicada a los proyectos realizados por este centro especializado donde la idea de pruebas exploratorio se aplica a la gestión de las pruebas funcionales y a la gestión de riesgos a través de los distintos ciclos de aplicación de los controles.

Para completar la información sobre el taller PRIS'07 se incluye una reseña del responsable del mismo, el profesor Javier Tuya de la Universidad de Oviedo, que glosa tanto la posición de esta iniciativa como las principales aportaciones de los participantes. La colaboración con iniciativas en esta línea de pruebas de software continuará en próximos eventos a celebrar en España con los que REICIS ya está acordando cauces de colaboración.

Finalmente, en la columna de Actualidad Invitada, es Pablo Pérez San José como gerente del Observatorio de la Seguridad de la Información en el Instituto Nacional de Tecnologías de la Comunicación (INTECO) quien realiza un repaso de las principales acciones que este organismo lleva a cabo como parte del Plan Avanza en relación con la calidad del software. Así se mencionan la creación del laboratorio nacional de calidad del software y la realización de estudios de diagnóstico en esta área entre otras iniciativas.

Luis Fernández Sanz

Generación e implementación de pruebas del sistema a partir de casos de uso

Javier J. Gutiérrez, María J. Escalona, Manuel Mejías, Arturo H. Torres, Jesús Torres
Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla
{javier, escalona,risoto, jtorres}@lsi.us.es

Abstract

Nowadays, there are several papers and book chapters which describe how to generate test cases from use cases. However, there is a lack of approaches showing how to implement those test cases into automate executable test cases. This paper introduces an architecture based on the UML testing profile and a set of steps to implement test objectives, which are defined as use case scenarios and operational variables.

Keywords: Functional testing, test automation, use case testing

Resumen

En la actualidad, existe una amplia cantidad de trabajos y capítulos de libros que proponen cómo obtener pruebas a partir de requisitos funcionales definidos como casos de uso. Sin embargo, existe una carencia de referencias que muestren cómo implementar dichas pruebas en código de pruebas automáticas. Este trabajo presenta una arquitectura, basada en el perfil de pruebas de UML, y un conjunto de pasos para la implementación en código ejecutable de pruebas de casos de uso definidas mediante escenarios y variables operacionales.

Palabras clave: Pruebas funcionales, automatización de las pruebas, pruebas de casos de uso.

1. Introducción

Un código sin fallos no tiene por qué resultar en un sistema sin fallos. Por ello, las pruebas del sistema cobran una gran importancia dentro de la etapa de pruebas. Las pruebas del sistema engloban tantos tipos de prueba como tipos de requisitos se puedan definir y probar con la ejecución del sistema o con la verificación de sus elementos. Habitualmente, incluye requisitos funcionales, de seguridad, de rendimiento, de fiabilidad, de accesibilidad, etc.

Este trabajo se centra en la prueba de requisitos funcionales definidos como casos de uso, es decir, en desarrollar un conjunto de casos de prueba que verifiquen si el comportamiento definido en dichos casos de uso ha sido correctamente implementado en el sistema. Al abordar la automatización de las pruebas de sistemas (y de otros tipos de pruebas), se pueden identificar tres niveles de automatización [1]. El primero es la

automatización de la generación de casos de prueba a partir de los requisitos, el segundo es la automatización de la ejecución de los casos de prueba y el tercero es la automatización de la comprobación de sus resultados. Existe un amplio número de artículos y capítulos de libros que describen cómo generar pruebas a partir de casos de uso (primer nivel de automatización). Sin embargo, varios trabajos comparativos y casos prácticos [2] [3] [4], exponen que la práctica totalidad de estas propuestas sólo generan descripciones de los escenarios y valores de prueba de una manera narrativa. La aportación original de este trabajo es un conjunto de pasos para la generación de código de prueba que permita llevar esos resultados al segundo y tercer nivel de automatización. Esto significa comprobar, de manera automática, si el sistema implementa el comportamiento definido en sus casos de uso. El resultado final será una prueba del sistema definida como código ejecutable con un conjunto de asertos para evaluar el resultado obtenido.

En este trabajo se van a utilizar dos artefactos diferentes, aunque complementarios, referentes a las pruebas del sistema. El primer artefacto se obtiene a partir de los casos de uso y contiene descripciones textuales de los escenarios a probar y sus valores de prueba (un ejemplo se muestra en la sección 2). El segundo artefacto será la implementación de dichos escenarios y valores de prueba en un fragmento de código ejecutable (como se muestra en la sección 4). Para evitar confusiones entre ambos artefactos, al primero de ellos se le llamará objetivo de prueba y, al segundo, caso de prueba.

La organización de este trabajo se describe a continuación. La sección 2 resume una serie de trabajos previos de los autores para la automatización de la generación de pruebas automáticas a partir de casos de uso. Los resultados obtenidos serán el punto de partida para la codificación de pruebas automáticas. Después, la sección 3 expone una arquitectura para la implementación de pruebas del sistema (a partir de los elementos definidos en el perfil de pruebas de UML [5]) y un conjunto de pasos para la redacción de pruebas como código ejecutable. La sección 4 muestra un caso práctico. Finalmente, la sección 5 describe otros trabajos relacionados, las conclusiones y los trabajos futuros.

2. Generación de objetivos de prueba a partir de casos de uso

A partir de los resultados de los estudios comparativos citados en la sección 1, se ha identificado que las dos técnicas más utilizadas para generar objetivos de prueba son la

definición de escenarios y la aplicación de técnicas de partición de dominios de variables (ambas documentadas en [6]). Estas dos técnicas se resumen en los siguientes párrafos.

La primera técnica, basada en los escenarios de un caso de uso, consiste en transformar el comportamiento de un caso de uso a un grafo y aplicar un criterio de recorrido para identificar distintos escenarios y utilizarlos como casos de prueba. Algunas propuestas que usan esta técnica son [7] [8] y [9].

La segunda técnica, basadas en técnicas de partición de dominios, consiste en identificar los puntos que pueden variar entre dos escenarios o realizaciones de un caso de uso (en [10] a estos puntos se les llaman variables operacionales), definir sus dominios, dividirlos en particiones, agrupando en cada partición aquellos valores para los que el caso de uso defina el mismo comportamiento, y, por último, construir una tabla con todas las combinaciones posibles de las particiones de cada variable operacional. Cada fila de la tabla, es decir, cada combinación de particiones, es un caso de prueba. Algunas propuestas que usan esta técnica son [10] o [11] para los puntos de variabilidad en casos de uso de familias de productos.

Existen también algunas técnicas complementarias a las dos técnicas anteriores. Por ejemplo, en la referencia [12] se proponen técnicas de análisis del lenguaje natural para la transformación de un caso de uso a grafo.

En este trabajo se definen los objetivos de prueba con la información de las dos técnicas anteriores. Un objetivo de prueba está compuesto de una secuencia de pasos, sin alternativa posible, de un conjunto de variables operacionales (cada una tomando valores de una partición concreta) y las pre y poscondiciones relevantes para dicho escenario. Un ejemplo de objetivo de prueba se muestra en la tabla 1. A continuación se resumen los pasos para obtener escenarios y combinaciones de particiones de un caso de uso automáticamente.

Para la generación de los escenarios de prueba, en primer lugar, se construye un diagrama de actividades a partir de los pasos de la secuencia principal y alternativas del caso de uso. En el diagrama de actividades, se identifican las acciones realizadas por el sistema y las acciones realizadas por los actores. Después, se realiza un recorrido mediante un criterio de suficiencia y cada camino del diagrama de actividades, será un escenario del caso de uso y, por tanto, una prueba potencial.

Se ha desarrollado una herramienta de código libre llamada ObjectGen, aún en fase experimental (www.lsi.us.es/~javierj/objectgen/), la cual permite obtener de manera

automática el diagrama de actividades y la lista de caminos a partir de un conjunto de casos de uso. Los algoritmos utilizados se describen con más detalle en [13]. Un ejemplo se muestra en la sección 4.1.

Para la generación de combinaciones de particiones se toma como punto de partida el diagrama de actividades obtenido en el párrafo anterior. A partir de él se identifican las variables operacionales presentes en el caso de uso. A partir de las variables operacionales, se aplica el método de Categoría-Partición (llamado por sus siglas: CPM) [14], considerando cada variable operacional como una categoría, para definir distintas particiones en los dominios de las variables operacionales tomando como base los nodos decisión del diagrama de actividades.

<table>
<tr><td>Name</td><td colspan="3">UC-01. 09</td></tr>
<tr><td>Test objective</td><td colspan="3">Main scenario</td></tr>
<tr><td>Priority</td><td colspan="3">No.</td></tr>
<tr><td>Initial state</td><td colspan="3">No.</td></tr>
<tr><td>Action sequence</td><td colspan="3"><table>
<tr><th>Id</th><th>Action description</th><th>Variables</th></tr>
<tr><td>1</td><td>The visitor selects the option for introduce a new link.</td><td>No.</td></tr>
<tr><td>2</td><td>The system recovers all the stored categories and it asks for the information of a link.</td><td>No.</td></tr>
<tr><td>D01</td><td>Not(there was an error recovering the categories) AND Not(there were not categories found)</td><td>V01 = P02
V02 = P02</td></tr>
<tr><td>D03</td><td>Not(cancel this operation)</td><td>V03 = P02</td></tr>
<tr><td>3</td><td>The visitor introduces the information of the new link.</td><td>V04 = P02</td></tr>
<tr><td>D04</td><td>Not(the link name, category or URL are empty)</td><td>No.</td></tr>
<tr><td>4</td><td>The system stores the new link.</td><td>V05 = P02</td></tr>
<tr><td>D05</td><td>Not(there is an error storing the link)</td><td></td></tr>
<tr><td></td><td>End.</td><td></td></tr>
</table></td></tr>
<tr><td>Final state</td><td colspan="3">A new links has been stored into the system.</td></tr>
<tr><td>Test values</td><td colspan="3"><table>
<tr><th>Instance</th><th>Partition</th><th>Sample value</th></tr>
<tr><td>V4_I1</td><td>P02</td><td>Nme: Test link
Category: default
URL: www.test.com</td></tr>
</table></td></tr>
<tr><td>Notes</td><td colspan="3">No.</td></tr>
</table>

Tabla 1. Ejemplo de objetivo de prueba

Para reducir el número de combinaciones y evitar combinaciones imposibles de realizar en la práctica, se identifican, a continuación, un conjunto de restricciones. Dichas

restricciones se expresan de la siguiente manera: *si una variable operacional V01 toma valor en su partición P01, entonces las variables V02…Vn no necesitan tomar valores.* Por último se calculan todas las posibles combinaciones teniendo en cuenta las restricciones identificadas.

También se ha desarrollado otra herramienta de código libre llamada ValueGen, también en fase experimental, la cuál permite obtener de manera automática un primer conjunto de variables operacionales, particiones, restricciones y combinaciones a partir de los casos de uso. Los algoritmos propuestos se describen en más detalle en [15]. Un ejemplo también se muestra en la sección 4.1.

La combinación de un camino en el diagrama de actividades con el conjunto de particiones de las cuáles las variables operacionales presentes en el camino deben tomar sus valores será un objetivo de prueba. El mismo objetivo de prueba que se utilizará en el caso práctico de la sección 4 se muestra en la tabla 1. En dicho objetivo, se puede apreciar cómo se incluye un escenario del caso de uso de la tabla 3 (sección 4), las variables operacionales y particiones necesarias, un ejemplo de valores de prueba, así como las precondiciones y poscondiciones presentes en el caso de uso. Este objetivo de prueba se muestra en inglés, dado que las dos herramientas utilizadas sólo soportan casos de uso en este idioma.

3. Implementación de pruebas del sistema

A continuación se describe cómo implementar los resultados obtenidos en la sección anterior. En la sección 3.1, se describe una arquitectura genérica para pruebas del sistema a partir de los elementos definidos en el perfil de pruebas de UML (llamado a partir de aquí por sus siglas en inglés: UMLTP). En la sección 3.2, se describe un conjunto de pasos para la implementación de pruebas funcionales del sistema, tomando como base la arquitectura definida en la sección 3.1.

3.1. Una arquitectura de pruebas del sistema

La arquitectura para la ejecución y comprobación automática de pruebas del sistema se muestran en la figura 1. Esta arquitectura está basada en la arquitectura xUnit [15], la cual es una de las más populares e implementada en la mayoría de las herramientas. A continuación, se describen brevemente los elementos de la arquitectura de prueba y su relación con los elementos definidos en el UMLTP cuando existe.

La clase *UserEmulator*, define el elemento que interactúa con el sistema bajo prueba utilizando las mismas interfaces que un actor. Si, por ejemplo, el sistema a prueba es una aplicación *web* (como en el caso práctico) la clase *UserEmulator* será capaz de interactuar con el navegador *web* para indicarle la URL qué tiene que visitar, rellenar formularios, pulsar enlaces, etc. Este elemento está estereotipado como un *Test Component*, ya que el UMLTP define de esta manera a los elementos auxiliares de los casos de prueba.

A partir de la interacción de dicha clase con el sistema, se obtendrán uno o varios resultados (clase *SystemOutput*); por ejemplo, en el caso del sistema web, se obtendrá código HTML. El perfil de pruebas de UML no define ningún elemento para representar los resultados obtenidos del sistema a prueba, por lo que se ha modelado con una clase sin estereotipar.

La clase *UserInterface* representa las interfaces externas del sistema bajo prueba. A partir de este tipo de elementos pueden definirse propiedades relevantes para las pruebas de cada interfaz mediante el estereotipo *sut* definido en el UMLTP.

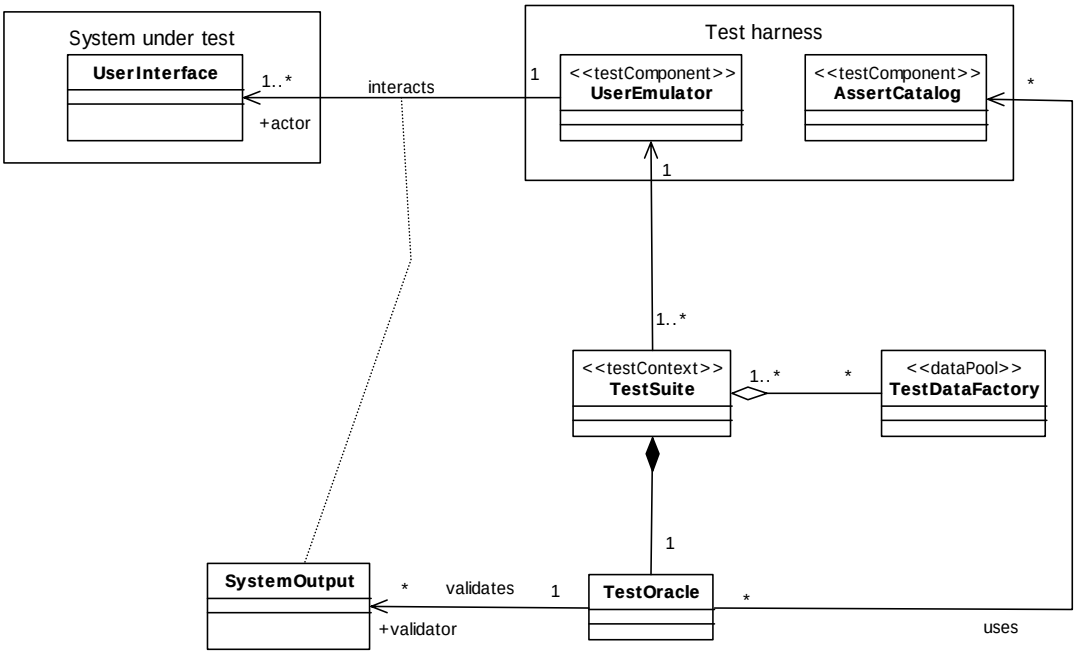

Figura 1. Arquitectura de pruebas

La clase *TestSuite*, estereotipada como un *Test Context* del perfil de pruebas de UML, representa un conjunto de casos de prueba (definidos en la siguiente sección). En el perfil de pruebas, todo *Test Context* tiene un elemento *Arbiter* y un elemento *Scheduler*, sin embargo, ambos elementos no se van a utilizar en esta propuesta (los proporcionará el *test harness*, sección 4.2), por lo que no aparecen en la figura 1.

La clase *AssertCatalog*, también estereotipada como *Test Component*, define la colección de asertos a disposición de los casos de prueba para determinar si el resultado obtenido del sistema a prueba (clase *SystemOutput*) es correcto o no. Tanto el *UserEmulator* como el *AssertCatalog* se han agrupado en un clasificador que representa el *Test Harness*, dado que estos dos elementos, suelen ser comunes para todas las prueba de diversos sistemas y existe gran variedad de ofertas en el mercado tanto de pago como libres y gratuitas.

El perfil de pruebas define el elemento *ValidationAction* (acciones de validación) que, como su nombre denota, permite indicar una operación concreta para determinar el resultado de un caso de prueba. Sin embargo, el perfil de prueba no define ningún elemento genérico para denotar todas las acciones de validación de un caso de prueba. Por este motivo se ha introducido dicho elemento en el marco de trabajo mediante la clase no estereotipada *TestOracle*. Esta clase sirve de contenedor de todas las acciones de validación (expresadas mediante los asertos del elemento *AssertCatalog* sobre el resultado obtenido y almacenado en el elemento *SystemOutput*) que determinarán el veredicto de los casos de prueba del contexto de prueba.

Por último, la clase *TestDataFactory* (estereotipada como *Data Pool* según el UMLTP) contendrá un conjunto de métodos, con el estereotipo *Data Selector*, para crear y/o seleccionar los distintos valores de prueba según las distintas particiones identificadas en las variables de los casos de uso.

Como se ha mencionado al principio, esta arquitectura es similar a la arquitectura xUnit, utilizada principalmente en pruebas unitarias. La principal diferencia estriba en que, en una prueba unitaria, la propia prueba invoca al código en ejecución, mientras que una prueba funcional del sistema necesita un mediador (el elemento *UserEmulator*) que sepa cómo manipular su interfaz externa. En la siguiente sección, se describen un conjunto de pasos para implementar los elementos de la figura 1 a partir de los objetivos de prueba.

3.2. Implementación de los casos de prueba

Para la implementación de los casos de prueba se van a aplicar los patrones *isolated test* y *test method* documentados en [15]. El primer patrón indica que cada caso de prueba debe ser independiente de los demás. Por ello, como se ha visto anteriormente, cada objetivo de prueba se codificará como un caso de prueba, ya que los objetivos de prueba no tiene dependencias entre ellos y cualquier objetivo de prueba puede verificarse independientemente de los demás. El segundo patrón indica que cada caso de prueba debe ser implementado como un método. Esto concuerda con la definición del UMLTP dónde un caso de prueba no es un elemento arquitectónico, por tanto no se ha definido en la sección anterior, sino la definición de un comportamiento como un método (estereotipado como *Test Case*) dentro de un elemento *Test Suite* (estereotipado como *Test Context*).

El comportamiento genérico de un caso de prueba que se adopta con mayor frecuencia se describe, entre otros trabajos, también [15] y se lista en la tabla 2. El segundo paso de la tabla 2 ha sido refinado en este trabajo con los pasos 2.1 y 2.2.

1. Invocación del *set up* del caso de pruebas.
2. Invocación del método de prueba
   2.1. Ejecución de una acción sobre el sistema.
   2.2. Comprobación del resultado de la acción (*opcional*).
3. Invocación del *tear down* del caso de pruebas.

Tabla 2. Comportamiento genérico de un caso de prueba

Como se puede ver en la tabla 2, cada método de prueba tiene asociados otros dos métodos. El primero, método *set-up*, establece el estado adecuado del sistema para la ejecución de la prueba. El segundo, *tear down*, restaura el estado original del sistema. Estos métodos también estarán definidos en el *Test Suite*.

Cada caso de uso tendrá asociado un elemento *TestSuite* (figura 1). Dicha *suite* contendrá los métodos de pruebas y métodos asociados de todos los escenarios de dicho caso de uso. A continuación se describe cómo implementar estos métodos y los demás elementos del modelo de la figura 1.

Como se ha visto, en cada uno de los pasos del objetivo de prueba se indica si es realizado por un actor o por el sistema a prueba. Esta información es relevante a la hora de la codificación de los métodos de prueba de la *suite*. Todos los pasos realizados por un actor se traducirán, en el código del caso de prueba, a una interacción entre el caso de prueba y el sistema a través del elemento *UserEmulator*.

El *test oracle* de un caso de prueba será el conjunto de acciones de validación obtenidas, principalmente, a partir de los pasos realizados por el sistema. Por ejemplo, en aquellos pasos en los que el sistema realice una petición de información u órdenes a los actores, se deberán definir asertos que permitan evaluar que todos los elementos de la interfaz son los correctos. En función de las poscondiciones del objetivo de prueba pueden añadirse asertos adicionales para verificar que el estado en que queda el sistema es el indicado en la poscondición. Las acciones de validación se implementan utilizando el catálogo de asertos proporcionado por el *test harness* sobre el resultado devuelto por el *UserEmulator*.

Además de los escenarios, se han utilizado variables operacionales para la definición de los valores de prueba necesarios (tabla 1). Antes de su implementación, estas variables se van a clasificar en tres tipos, cada uno de los cuáles se implementará de manera distinta.

El primer tipo lo componen aquellas variables operacionales que indican un suministro de información al sistema por parte de un actor. Para cada variable de este tipo se definirá una nueva clase cuyos objetos contendrán los distintos valores de prueba para dicha variable. Estas clases se codificarán aplicando el patrón *Value Object* [15], por lo que tendrán un método constructor parametrizado para establecer el valor de sus atributos y un conjunto de métodos *set* para acceder al valor de dichos atributos. Además, para cada partición del dominio identificada, el elemento *TestDataFactory* tendrá, al menos, un método estereotipado como *Data Selector* que devolverá un valor de prueba perteneciente a dicha partición. Un ejemplo de una variable operacional de este tipo se muestra en el caso práctico, en la tabla 6 (a).

El segundo tipo lo componen aquellas variables operacionales que indican una selección entre varias opciones que un actor tiene disponible. En este caso, no tiene sentido implementar estas variables como métodos del *TestDataFactory*. En su lugar, dicha selección se implementará directamente como parte del código que implementa la

interacción entre el actor y el sistema. Un ejemplo de una variable operacional de este tipo se muestra en el caso práctico en la tabla 6 (a).

El tercer tipo lo componen aquellas variables operacionales que indican un estado del sistema, bien una información que el sistema almacena o una configuración concreta. Para implementar el método de *set up* del caso de prueba, se debe escribir el código necesario para establecer adecuadamente el valor de las variables operacionales que describen los estados del sistema, o bien comprobar que dichos valores son los adecuados. De manera análoga, el método *tear down* debe restaurar dichos valores a sus estados originales. Además, dicho método debe eliminar, si es procedente, la información introducida por el caso de prueba en el sistema durante la ejecución del caso de prueba. Varios ejemplos de variables operacionales de este tipo se muestran en el caso práctico también en la tabla 6 (a).

A continuación se muestra un caso práctico de todo lo expuesto en las secciones anteriores.

4. Un caso práctico

Como caso práctico, para ilustrar todo lo visto, se ha tomado una aplicación web para la gestión de un catálogo de enlaces on-line. Esta aplicación permite realizar consultas a partir de los enlaces almacenados, o bien añadir nuevos enlaces al catálogo. En primer lugar se aplicará lo visto en la sección 2 para obtener un conjunto de objetivos de prueba a partir de un caso de uso (sección 4.1). Después, se definen las características del *Test harness* utilizado (sección 4.2). Finalmente, se aplica lo visto en la sección 3 para implementar un caso de prueba a partir de un objetivo de prueba (sección 4.3). Los artefactos del sistema bajo prueba se han definido en inglés, ya que el español no está soportado por las herramientas ObjectGen y ValugeGen.

4.1. Objetivos de prueba

El caso de uso de la tabla 3, describe la introducción de un nuevo enlace en el sistema. Como complemento, se muestra también el requisito de almacenamiento de información que describe la información manejada por cada enlace (tabla 4). Los patrones usados se han tomado de la metodología de desarrollo NDT [16] [17].

A partir del caso de uso, y de manera automática, se han generado un diagrama de actividades (figura 2).

Name	UC-01. Add new link		
Precondition	No		
1Main sequence	1	The user selects the option for introducing a new link.	
	2	The system recovers all the stored categories and it asks for the information of a link.	
	3	The user introduces the information of the new link.	
	4	The system stores the new link.	
Alternatives	3.1	At any time, the user can cancel this operation, then this use case ends.	
Errors	2.1	If there was an error recovering the categories, then the system shows an error message and this use case ends.	
	2.2	If there were not categories found, then the system shows and error message and this use case ends.	
	3.2	If the link name, category or link URL is empty, then the system shows an error message with the result of repeat step 2.	
	4.1	If there is an error storing the link, then the system shows an error message and this use case ends.	
Post condition	A new link is stored.		

Tabla 3. Caso de uso a prueba

Name	SR-01. Link.	
Specific data	*Name*	*Nature*
	Identifier	Integer
	Name	String
	Category	Integer
	URL	String
	Description	String
	Approved	Boolean
	Adding date	Date
Restrictions	The identifier must be unique. Name, category, URL and approved are mandatory Default value for approved is false (0) and for date is the actual date.	

Tabla 4. Requisito de almacenamiento complementario al caso de uso

Dado que el caso de uso presenta bucles no acotados, con un número potencialmente infinito de repeticiones (el número de veces que el actor introduce un enlace erróneo), el criterio de cobertura elegido consiste en obtener todos los caminos posibles para una repetición de ninguna o una vez de cada uno de los bucles.

Todos los escenarios obtenidos con este criterio, y traducidos al español, se listan en la tabla 5 (a). Para este caso práctico, seleccionamos el escenario 09 para su implementación. Este escenario se describe en detalle en la tabla 5 (b) (dado que el caso de uso se ha redactado en inglés, su escenario principal también se muestra en inglés).

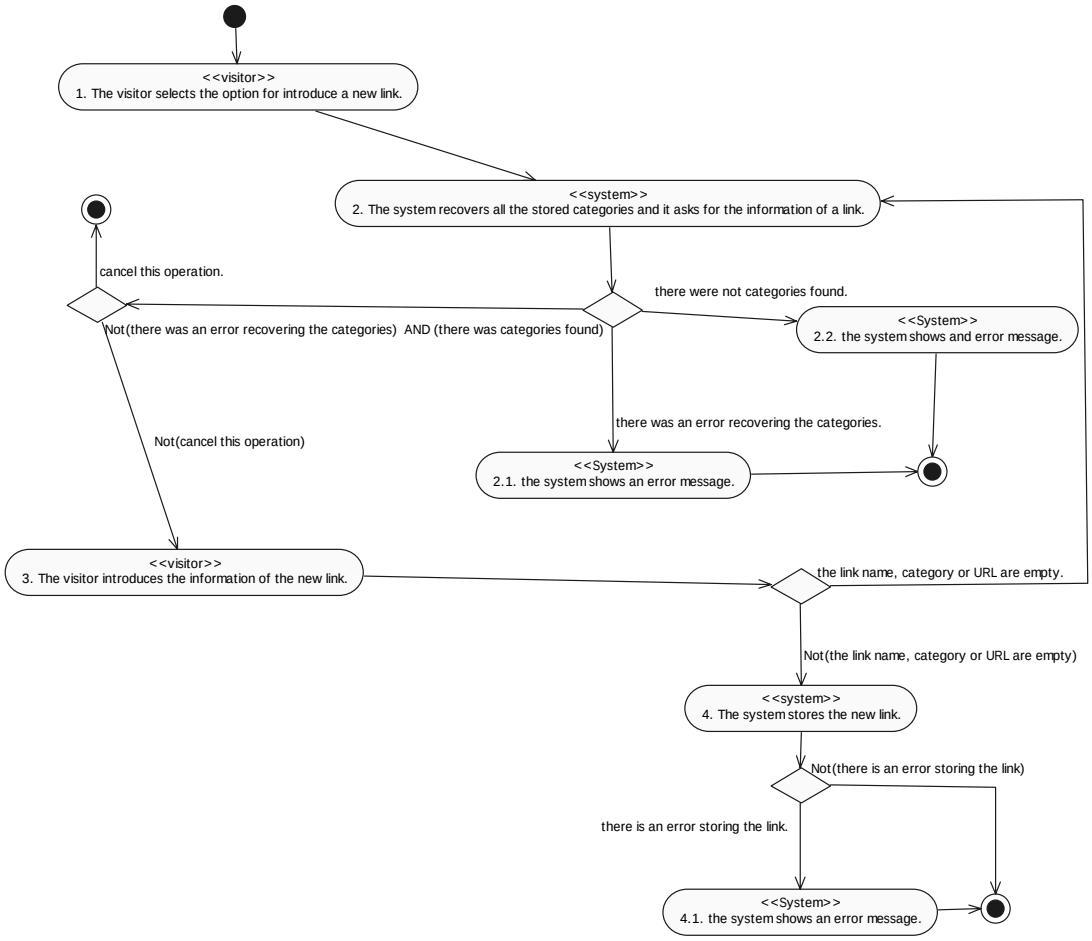

Figura 2. Diagrama de actividades del caso de uso.

A continuación, se ha aplicado CPM tal y como se ha descrito en la sección 2. Todas las variables operacionales encontradas automáticamente se enumeran en la tabla 6 (a) y las sus particiones se enumeran en la tabla 6 (b), ambas traducidas al español. En este caso, no se ha continuado refinando el conjunto de particiones aunque para algunas variables, como V04, sí podrían identificarse particiones adicionales para, por ejemplo, distinguir entre los distintos motivos por los que un enlace puede ser erróneo.

Todas las combinaciones válidas calculadas mediante la herramienta ValueGen se muestran en la tabla 7. Un "*" significa que dicha variable no toma valor en esa combinación (a causa de las restricciones identificadas como se ha mostrado en la sección

2). Como algunas variables y particiones (en concreto cuando V04 = P01) provocan un bucle y que se vuelvan a evaluar algunas variables por segunda vez, se ha identificado esta segunda instancia, añadiendo "_2" al final del nombre de la variable en la tabla 7.

Escenario	Descripción
01	Aparece un error recuperando las categorías.
02	El usuario cancela la operación.
03	El usuario introduce un enlace incorrecto y, después, aparece un error recuperando las categorías.
04	El usuario introduce un enlace incorrecto y, después, el usuario cancela la operación.
05	El usuario introduce un enlace incorrecto y, después, aparece un error al almacenar el enlace.
06	El usuario introduce un enlace incorrecto y, después, el usuario introduce un enlace correcto.
07	El usuario introduce un enlace incorrecto y, después, el sistema no encuentra ninguna categoría.
08	Aparece un error al almacenar el enlace.
09	*El usuario introduce un enlace correcto (camino principal).*
10	El sistema no encuentra ninguna categoría.

(a)

Paso	Descripción
01	The user selects the option for introduce a new link.
02	The system recovers all the stored categories and it asks for the information of a link.
03	Not(there was an error recovering the categories) AND Not(there were not categories found)
04	Not(cancel this operation)
05	The user introduces the information of the new link.
06	Not(the link name, category or URL are empty)
07	The system stores the new link.
08	Not(there is an error storing the link)

(b)

Tabla 5. (a) Escenarios del caso de uso y (b) escenario principal

Variable	Descripción	Tipo
V01	Error al recuperar categorías.	Sistema.
V02	Categorías encontradas.	Sistema.
V03	Opción del usuario	Actor.
V04	Datos del enlace	Información.
V05	Error al almacenar el enlace,	Sistema..

(a)

Variable	Particiones
V01	P01: Ocurre un error. P02: No ocurre un error.
V02	P01: No se encontraron categorías. P02: Sí se encontraron categorías.
V03	P01: Cancela la operación. P02: No cancela la operación.
V04	P01: El nombre, categoría o URL están vacías. P02: El enlace es correcto.
V05	P01: Error almacenando el enlace. P02: No ocurre un error.

(b)

Tabla 6. (a) Variables y (b) particiones identificadas para el caso de uso

Id	Combinación
1	V01:P01 V02:* V03:* V04:* V05:*
2	V01:P02 V02:P01 V03:* V04:* V05:*
3	V01:P02 V02:P02 V03:P01 V04:* V05:*
4	V01:P02 V02:P02 V03:P02 V04:P01 V01_2:P01 V02_2:* V03_2:* V04_2:* V05:*
5	V01:P02 V02:P02 V03:P02 V04:P01 V01_2:P02 V02_2:P01 V03_2:* V04_2:* V05:*
6	V01:P02 V02:P02 V03:P02 V04:P01 V01_2:P02 V02_2:P02 V03_2:P01 V04_2:* V05:*
7	V01:P02 V02:P02 V03:P02 V04:P01 V01_2:P02 V02_2:P02 V03_2:P02 V04_2:P02 V05:P01
8	V01:P02 V02:P02 V03:P02 V04:P01 V01_2:P02 V02_2:P02 V03_2:P02 V04_2:P02 V05:P02
9	V01:P02 V02:P02 V03:P02 V04:P02 V05:P01
10	V01:P02 V02:P02 V03:P02 V04:P02 V05:P02

Tabla 7. Combinaciones válidas para las variables y particiones identificadas

Para la implementación del escenario de éxito (escenario 09), todas las variables operacionales deben tener un valor perteneciente a las particiones P02, lo cuál coincide con la combinación 10.

La combinación de los pasos del escenario 09 con sus variables operacionales y particiones, así como la precondición y poscondición del caso de uso, dan lugar al objetivo de prueba mostrado en la tabla 1. En la siguiente sección, se definen los elementos pertenecientes al *test harness* usados en este caso práctico.

4.2. *Test harness*

Tal y como se describió en la figura 1, el *test harness* permite simular el comportamiento del usuario y ofrecer un conjunto de asertos para evaluar el resultado obtenido.

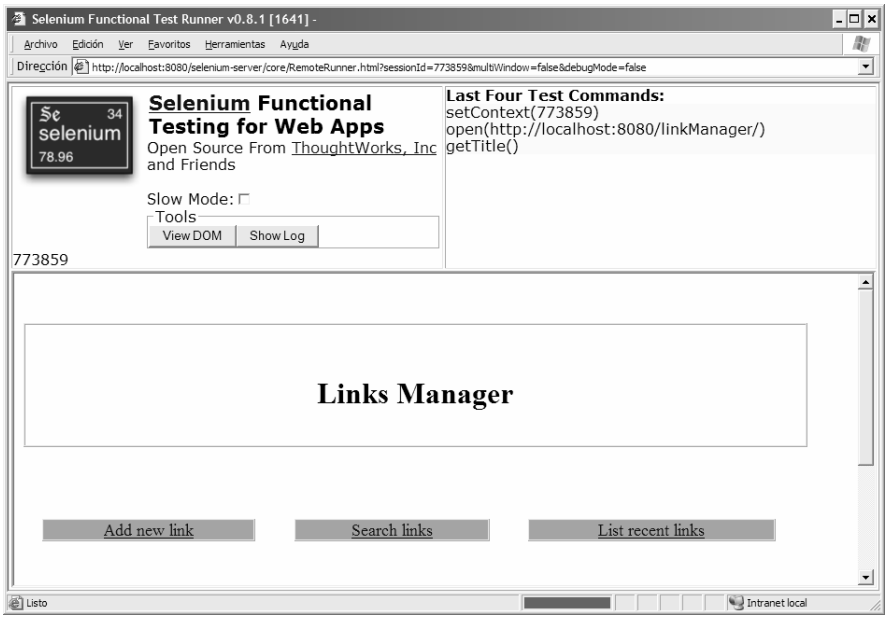

Figura 3. Ejecución del caso de prueba con la herramienta Selenium.

En este caso práctico, al ser el sistema bajo prueba una aplicación *web*, es necesario que el *test harness* sea capa de comunicarse con el navegador *web* y pueda ejecutar asertos sobre el código HTML recibido como respuesta. La herramienta elegida ha sido Selenium (www.openqa.org/selenium), gratuita y de código abierto. Como se puede ver el la figura 3, Selenium permite abrir un navegador *web* e interactuar con él de la misma manera que lo haría un actor humano. Esta herramienta está basada en la popular herramienta JUnit, por lo que Selenium trabaja con la misma arquitectura, incorpora el mismo conjunto de asertos que JUnit y, además, funciones adicionales para acceder a los resultados visualizados en el navegador *web*. Además, Selenium permite escribir las pruebas en varios lenguajes, aunque en este caso práctico se ha utilizado el lenguaje Java.

4.3. Implementación de un caso de prueba

En primer lugar se ha implementado el elemento *TestDataFactory* y los valores de prueba. De la tabla 6 (a), sólo la variable V04: Datos del enlace, hace referencia a una información suministrada desde el exterior al sistema durante la ejecución de la prueba (como se ha visto en la sección 3.2). Se ha desarrollado una clase (clase *Link* en la figura 4), aplicando el patrón *value object* resumido con anterioridad, para representar los diferentes enlaces. Los atributos han sido identificados a partir del requisito de almacenamiento de la tabla 4. Por cada partición posible, se ha añadido un método estático al elemento *TestDataFactory* para obtener un objeto enlace con valores adecuados a su categoría. Como se mencionó en la sección 4.2, sólo se han tenido en cuenta las dos particiones principales: enlaces correctos e incorrectos (tabla 6 (b)), sin entrar en más subdivisiones, por lo que sólo hay dos métodos *Data Selector*.

Después, se ha definido el *test suite* (clase *TestUC01* en la figura 4) con un método de pruebas, un método *set-up* y un método *tear-down* por cada escenario. En la figura 4, se muestra sólo los métodos correspondientes al escenario de la secuencia principal del caso de uso (escenario 09).

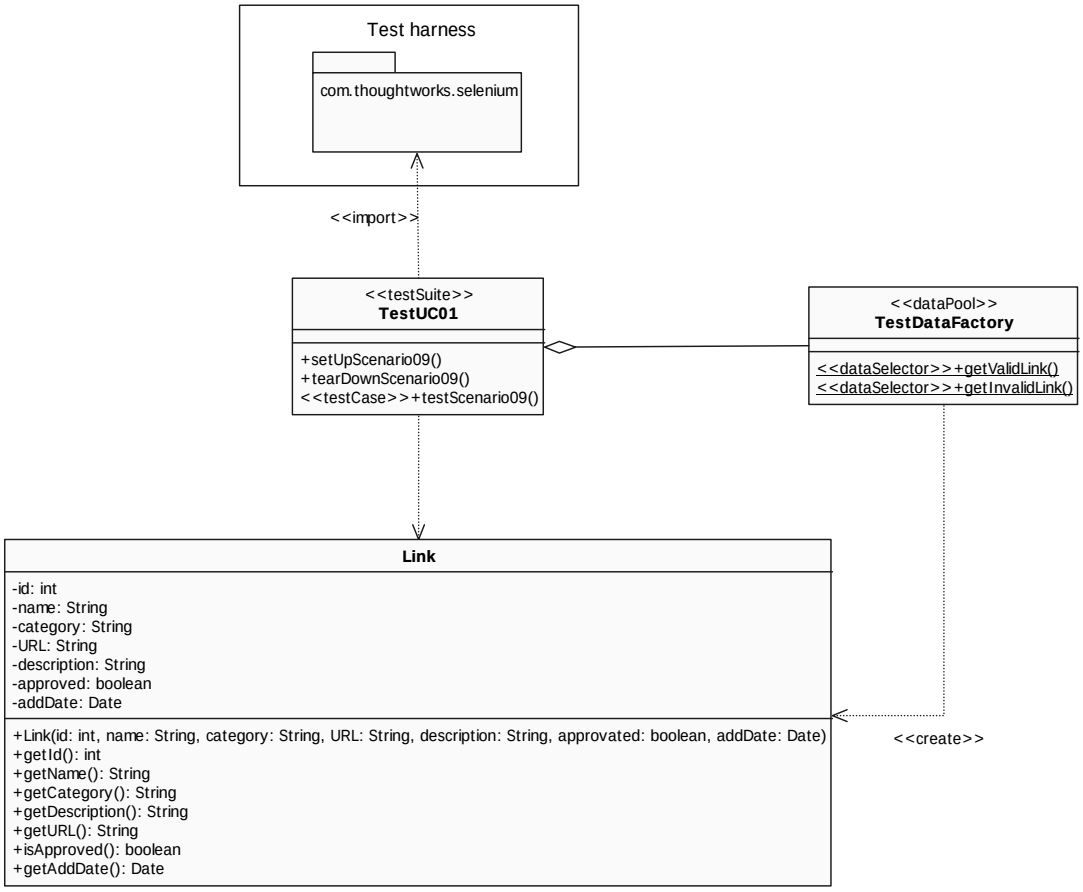

Figura 4. Implementación del caso de pruebas

A continuación, se toman todos los pasos ejecutados por el actor humano y se traducen a código Java para la herramienta Selenium. Dicha traducción se ha realizado a mano y el código resultante se muestra en la tabla 8. La referencia *s* apunta a una instancia del *test harness* que, para este caso concreto, es una instancia de una clase llamada *Selenium*. El método *click* realiza una pulsación sobre el elemento indicado (bien en el enlace *AddNewLink*, bien en el botón *submit* del formulario), y el método *type*, rellena un control de un formulario.

Como se mencionó en la sección 4.2, la variable V03: Opción del usuario, se implementa como parte del código del caso de prueba (última línea en cursiva que aparece en la tabla 8).

Por las características específicas de Selenium, es necesario añadir esperas a que la página se cargue antes de continuar mediante el método *waitForPageLoad(tiempo)*. Esto

significa que, en la implementación del paso 1, la prueba esperará como máximo 5 segundos a que la página se cargue, si no, dará la prueba como no superada.

Paso:	**Código:**
01: The user selects the option for introduce a new link.	`s.click("AddNewLink");` `s.waitForPageToLoad("5000");`
05: The user introduces the information of the new link.	`Link l = TestDataPool.getValidLink();` `s.type("name", l.getName());` `s.type("URL", l.getURL());` `s.type("description", l.getDescription());` *`s.click("addEvent_0");`*

Tabla 8. Traducción a código ejecutable de los pasos realizados por el usuario en el escenario principal

Después, se escriben los asertos a partir de los pasos que realiza el sistema. El código resultante para la herramienta Selenium se muestra en la tabla 9. En el primer conjunto de asertos (paso 02) se verifica el título de la página obtenida como respuesta y que estén presentes todos los controles necesarios del formulario. En el segundo conjunto de asertos (paso 07), se verifica el título de la página obtenida y que no esté presente el mensaje de error. Además, en el paso 07, se ha incluido un aserto adicional que verifique la poscondición, es decir, que compruebe que el enlace está correctamente almacenado en el sistema tal. Para ello se ha añadido un método auxiliar *isLinkStored* (última línea del paso 07, tabla 9), el cuál indica si el enlace suministrado como parámetro ha sido almacenado por el sistema.

Paso:	**Código:**
02: The system recovers all the stored categories and it asks for the information of a link.	`assertEquals("Add new link form", sel.getTitle());` `assertTrue(s.isTextPresent("Name*:"));` `assertTrue(s.isElementPresent("addEvent_name"));` `assertTrue(s.isTextPresent("Category*:"));` `assertTrue(s.isElementPresent("addEvent_category"));` `assertTrue(s.isTextPresent("URL*:"));` `assertTrue(s.isElementPresent("addEvent_URL"));` `assertTrue(s.isTextPresent("Description:"));` `assertTrue(s.isElementPresent("addEvent_description"));` `assertTrue(s.isTextPresent("Date:"));` `assertTrue(s.isElementPresent("addEvent_date"));` `assertTrue(s.isElementPresent("addEvent_0"));`
07: The system stores the new link.	`assertEquals("Links manager", s.getTitle());` `assertFalse(s.isTextPresent("Error storing new link"));` `assertTrue(isLinkStored(TestDataPool.getValidLink()));`

Tabla 9. Traducción a código ejecutable del *test oracle* para el escenario principal

Después, el código de las tablas 8 y 9 se reordena siguiendo la secuencia de pasos del escenario de la tabla 5 (b). Los pasos 03, 04 y 06 no aparecen en ninguna de las dos tablas

porque no son acciones de la prueba sino definiciones de variables y particiones y, por tanto, ya han sido tenidas en cuenta al elegir las particiones adecuadas.

Finalmente, la implementación del método de *set-up* consiste en comprobar que todas las variables operacionales del sistema, mostradas en la tabla 6 (a), tengan un valor de la partición P02. Es decir, comprobar que hay categorías almacenadas en el sistema y que no hay ninguna circunstancia que ocasione un error al recuperar las categorías o insertar el nuevo enlace. La implementación del método de *tear down*, consiste en la restauración del conjunto original de enlaces almacenado en el sistema.

El código resultado puede descargarse de la misma página que hospeda las herramientas ObjectGen y ValueGen.

5. Conclusiones

En este trabajo se ha mostrado un proceso para implementar casos de prueba a partir de objetivos de prueba para casos de uso. Existe un número muy limitado de trabajos que aborden este proceso desde una perspectiva similar. Por ello, se han adaptado varias ideas (arquitectura xUnit, patrones, etc.) de la codificación de pruebas unitarias. Otros trabajos relacionados con la generación de pruebas ejecutables se citan en los siguientes párrafos.

En [15] se pueden encontrar distintos patrones para la implementación de casos de prueba unitaria. Varias de sus ideas se han utilizado en este trabajo. En [18] se muestra un ejemplo de generación automática de código de pruebas para pruebas unitarias, basado en técnicas de reflexión aplicadas sobre el código original. En este caso, los objetivos de prueba se definen cómo combinaciones de valores de prueba a verificar por las pruebas generadas. En [19] se describe un caso práctico sobre la prueba de sistemas móviles a través de GUI utilizando como punto de partida modelos y lenguajes específicos de dominio. En concreto, para dicho caso práctico se describió un lenguaje de modelado específico, el cuál se implementó con posterioridad mediante un conjunto de eventos de la interfaz gráfica. Siguiendo esta línea, una posible extensión del trabajo presentado en este artículo consistiría en definir un *script* de prueba en un lenguaje independiente que después pueda ser implementado en distintas herramientas. Un ejemplo preliminar de esto se puede encontrar en [20].

Como se ha mencionado a lo largo de este trabajo, se ha conseguido automatizar la generación de pruebas mediante dos herramientas (el primer nivel de automatización mencionado en la introducción). Para la generación automática de código ejecutable, como se ha visto en el caso práctico, necesita tener muchos datos específicos de la aplicación y de la interfaz, sino definidos de una manera procesable automáticamente y enriquecerlos con una semántica para que el sistema sepa lo que son. Nuestros próximos trabajos apuestan por el uso de modelos de diseño, asociados a los casos de uso y con la convención de nombres, desde los requisitos hasta la implementación, para poder aplicar generación automática.

Agradecimientos

Este trabajo ha sido realizado en el marco del Ministerio de Ciencia y Educación de España, bajo el Programa de Investigación, Desarrollo e Innovación, proyecto QSimTec (TIN2007-67843-C06-03) y REPRIS (TIN2005-24792-E).

Referencias

[1] Meudec C. "ATGen: Automatic Test Data Generation Using Constraint Logic Programming and Symbolic Execution". *ACM SIGSOFT Software Engineering Notes.* Vol. 23, nº 2, pp. 53-62, 1998.

[2] Denger, C. Medina M. *Test Case Derived from Requirement Specifications*. Fraunhofer IESE Report, 2003.

[3] Gutiérrez, J.J., Escalona M.J., Mejías M., Torres, J. "Generation of test cases from functional requirements. A survey". *4º Workshop on System Testing and Validation*, Alemania, 2006.

[4] Roubtsov S. y Heck P. "Use Case-Based Acceptance Testing of a Large Industrial System: Approach and Experience Report". *Testing: Academic & Industrial Conference (TAIC PART*). Windsor, Reino Unido, 2006.

[5] Object Management Group. *The UML Testing Profile*. Disponible en: www.omg.org . 2003.

[6] Díaz E. Blanco R. Tuya J. "Los Métodos de Generación de Casos de Prueba y su Automatización". En: Tuya J. Ramos I. Dolado J. (editores). *Técnicas cuantitativas para la Gestión en la Ingeniería del Software*. Editorial Netbiblo, 2007.

[7] Fernández L. Lara P. Escribano J.J. "Gestión Cuantitativa del Diseño de Casos de Prueba Mediante Casos de Uso". En: Tuya J. Ramos I. Dolado J. (editores). *Técnicas cuantitativas para la Gestión en la Ingeniería del Software*.. Editorial Netbiblo. 2007.

[8] Naresh, A., "Testing From Use Cases Using Path Analysis Technique", *International Conference On Software Testing Analysis & Review*, EE.UU, 2002.

[9] Ruder A., "UML-based Test Generation and Execution". *Rückblick Meeting*, Berlin. 2004.

[10] Binder, R.V. *Testing Object-Oriented Systems*. Addison Wesley. 1999.

[11] Bertolino, A., Gnesi, S. "PLUTO: A Test Methodology for Product Families2. *Lecture Notes in Computer Science*. Springer-Verlag Heidelberg. 3014 / 2004. pp 181-197. 2004.

[12] Boddu R., Guo L. y Mukhopadhyay S., "RETNA: From Requirements to Testing in Natural Way", *12th IEEE International Requirements Engineering RE'04*. 2004.

[13] Gutiérrez J.J. Escalona M.J. Mejías M. Torres J., "Modelos y algoritmos para la generación de objetivos de prueba". *XIII Jornadas sobre Ingeniería del Software y Bases de Datos JISBD*. Sitges, España, 2006.

[14] Ostrand T. J. y Balcer M. J., "The Category-Partition Method". *Communications of the ACM*, pp 676-686, 1988.

[15] Beck K., *Test-Driven Development: By Example*, Addison-Wesley, 2002.

[16] Escalona M.J., *Models and Techniques for the Specification and Analysis of Navigation in Software Systems*. Ph. European Thesis. University of Seville. Sevilla, España, 2004.

[17] Escalona M.J., Gutiérrez J.J., Villadiego D., León A. y Torres A.H., "Practical Experiences in Web Engineering". 1*5th International Conference On Information Systems Development*. Budapest, Hungria, 2006.

[18] Polo M., Tendero S.y Piattini M, "Integrating Techniques and Tools for Testing Automation", *Journal of Software Testing, Verification and Reliability*, vol.17, pp. 3-39, 2006.

[19] Katare M. et-al. "Towards Deploying Model-Based Testing with a Domain-Specific Modeling Approach". *Testing: Academic & Industrial Conference (TAIC PART*). Windsor, Reino Unido, 2006.

[20] Gutiérrez J.J., Escalona M.J., Mejías M. y Reina A.M., "Modelos de pruebas para pruebas del sistema". *Taller de Desarrollo de Software Dirigido por Modelos. XIII Jornadas sobre Ingeniería del Software y Bases de Datos JISBD*, Sitges, España, 2006.

Estrategia de gestión de las pruebas funcionales en el Centro de Ensayos de Software

Beatriz Pérez Lamancha
Centro de Ensayos de Software
bperez@fing.edu.uy

Abstract

This article presents a strategy for managing the functional testing of a software product. The strategy defines testing scope and agenda based on the product risk analysis, combining test cases design with exploratory testing. Test cycles are defined according to the product development plan and an overall test plan is prepared based on its functionalities, which is reviewed and refined before each cycle starts. Exploratory testing has a fundamental role in this approach. On the one hand, it helps to mitigate possible errors in the product risk analysis as leaving out important business functionalities. The product risk analysis must be reviewed when a critical bug for the business is found during an exploratory testing session, which was not detected by the execution of designed test cases. On the other hand, exploratory testing complements the test cases when the available time is not enough to design test cases to cover all the functionalities in the test cycle.

KeyWords: Software engineering, testing, test management, functional testing.

Resumen

Se presenta en este artículo una estrategia para la gestión de las pruebas funcionales de un producto de software. La estrategia define el alcance y la agenda de las pruebas a partir del análisis de riesgo del producto, combinando los casos de prueba con diseño previo y el testing exploratorio. Se definen los ciclos de prueba en función del plan de desarrollo del producto y se elabora una planificación global a partir de sus funcionalidades, la que es revisada y refinada al comenzar cada ciclo de prueba. El testing exploratorio cumple un papel fundamental en la estrategia. Por un lado, ayuda a mitigar la posibilidad de equivocarse al realizar el análisis de riesgo del producto, dejando de lado funcionalidades importantes para el negocio. Si en una sesión de testing exploratorio se encuentra un incidente crítico para el negocio que no fue detectado por los casos de prueba diseñados, el análisis de riesgo debe ser revisado. Por otro lado, complementa la prueba con diseño previo cuando no se dispone del tiempo suficiente en un ciclo como para generar los casos de prueba que cubran las funcionalidades requeridas.

Palabras clave: Ingeniería de software, pruebas, gestión de las pruebas, Pruebas Funcionales.

1. Introducción

El objetivo de las pruebas funcionales es validar si el comportamiento observado del software cumple o no con sus especificaciones. La prueba exhaustiva del producto requiere ejercitar todos los caminos posibles del mismo, incluso para un programa pequeño, el tiempo requerido para esto es excesivo. Esto impacta directamente en la economía de las pruebas, ya que se deberán realizar suposiciones sobre el comportamiento del programa y la forma en que se diseñan los casos de prueba para el mismo. El objetivo es maximizar la producción de las pruebas, esto es, maximizar el número de los errores encontrados por un número finito de los casos de prueba [1]. La técnica con que se seleccionan los casos de prueba es uno de los principales supuestos a definir.

Existen distintos procesos y metodologías para las pruebas funcionales de software, agrupan en general las actividades referentes a las pruebas en las etapas de planificación, diseño y de ejecución de las pruebas. Durante la planificación de las pruebas se decide qué se probará y con qué profundidad, en la etapa de diseño, la especificación de requisitos se analiza para derivar los casos de prueba y en la etapa de ejecución se ejecutan los casos de prueba diseñados previamente, se compara el resultado real con el esperado y se reportan los resultados.

Se presenta en este trabajo una estrategia para la gestión de las pruebas funcionales que define el alcance y la agenda de las pruebas del proyecto en función del análisis de riesgo del producto, combinando los casos de prueba derivados utilizando técnicas de caja negra y el testing exploratorio. De esta forma se retroalimentan los casos de pruebas diseñados con los resultados del testing exploratorio. Esta estrategia de gestión es la utilizada para realizar servicios de prueba independiente en el Centro de Ensayos de Software [1] (CES).

El resto del artículo está organizado como sigue. La siguiente sección describe brevemente los principales conceptos referentes a las pruebas funcionales. La sección 3 presenta el Centro de Ensayos de Software y el proceso de pruebas funcionales que sigue en sus proyectos. La sección 4 describe la estrategia de planificación de las pruebas funcionales. Finalmente, la sección 5 presenta las conclusiones y trabajo futuro.

[1] Centro de Ensayos de Software, Montevideo, Uruguay. http://www.ces.com.uy

2. Pruebas funcionales

En esta sección se definen los principales términos referidos en este trabajo. Se define prueba funcional, prueba de regresión y *testing* exploratorio.

La prueba funcional es conocida también como basado en la especificación o de caja negra. El objetivo de la prueba funcional es validar si el comportamiento observado del software cumple o no con sus especificaciones. La prueba funcional toma el punto de vista del usuario [2]. Las funciones son probadas ingresando las entradas y examinando las salidas, la estructura interna del programa raramente es considerada [3]. Para realizar pruebas funcionales, la especificación se analiza para derivar los casos de prueba. Técnicas como partición de equivalencia, análisis del valor límite, grafo causa-efecto y conjetura de errores son especialmente pertinentes para las pruebas funcionales. Se deben considerar condiciones inválidas e inesperadas de la entrada y tener en cuenta que la definición del resultado esperado es una parte vital de un caso de la prueba. El propósito de la prueba funcional es mostrar discrepancias con la especificación y no demostrar que el programa cumple con su especificación [1].

Las pruebas de regresión tienen como objetivo verificar que no ocurrió una regresión en la calidad del producto luego de una modificación, asegurando que los cambios no introducen un comportamiento no deseado o errores adicionales. Implican la reejecución de alguna o todas las pruebas realizadas anteriormente [4].

El término "*testing* exploratorio" fue introducido por Cem Kaner [5], se trata de ejecutar las pruebas a medida que se piensa en ellas, utilizando muy poco tiempo en preparar o explicar las pruebas, confiando en los instintos. El *testing* exploratorio se define como el aprendizaje, el diseño de casos de prueba y la ejecución de las pruebas en forma simultánea. En otras palabras, es cualquier prueba en la cual quien prueba controla activamente el diseño de las pruebas mientras las pruebas se ejecutan, y utiliza la información obtenida mientras prueba para diseñar nuevas y mejores pruebas [5]. En el *testing* exploratorio siempre se debe tomar nota de lo que se hizo y lo que sucedió [3]. Los resultados obtenidos no son necesariamente diferentes de aquellos obtenidos de la prueba con diseño previo y ambos enfoques para las pruebas son compatibles [5].

El *testing* exploratorio puede ser realizado en cualquier situación donde no sea obvio cual es la próxima prueba que se debe realizar. También cuando se requiere obtener

retroalimentación rápida de cierto producto o funcionalidad, se necesita aprender el producto rápidamente, se quiere investigar y aislar un defecto en particular, se quiere investigar el estado de un riesgo particular, o se quiere evaluar la necesidad de diseñar pruebas para determinada área. Una estrategia básica para realizar *testing* exploratorio es tener un plan de ataque general, pero permitirse desviarse de él por periodos cortos de tiempo. Cem Kaner llama a esto el principio del "tour bus", las personas bajan del bus y conocen los alrededores. La clave es no perderse el tour entero [5]. En general, el *testing* solamente exploratorio puede funcionar para *testers* con mucha experiencia. Como ventaja se encuentra que es barato y rápido, como contra que no se tienen medidas de cubrimiento y no deja documentación [4].

3. Centro de Ensayos de Software

El Centro de Ensayos de Software (CES) es un emprendimiento conjunto de la Universidad de la República de Uruguay (UdelaR) y de la Cámara Uruguaya de Tecnologías de la Información (CUTI), entidad que agrupa a la mayoría de las empresas productoras de software de Uruguay. Tiene por objetivo brindar servicios especializados de testing a la industria de tecnologías de la información (TI), para mejorar su capacidad productiva en cuanto a calidad, diversidad de plataformas e innovación de sus productos.

Los servicios que ofrece el CES incluyen

- Servicios de prueba independiente: Planificar, diseñar, coordinar y ejecutar pruebas de productos de software de manera efectiva y controlada, definiendo claramente el contexto y los objetivos.
- Consultoría: Asesorar a las organizaciones en la mejora de los procesos de prueba, definición de estrategias y automatización de las pruebas. Colaborar en la creación y consolidación de sus áreas de prueba.
- Capacitación: Elaborar e impartir programas de capacitación en la disciplina de testing según las necesidades de cada organización.

El CES se compone de dos laboratorios: el Laboratorio de Testing Funcional enfocado en la evaluación de productos desde el punto de vista funcional y el Laboratorio de Ensayos de Plataformas, donde se realizan pruebas de desempeño y se asiste a la

industria para resolver problemas de funcionamiento en arquitecturas de hardware y software complejas.

En la Figura 1 se muestra la cantidad de proyectos en los que ha trabajado el CES desde su creación, distinguiendo por tipo de proyecto hasta junio de 2007.

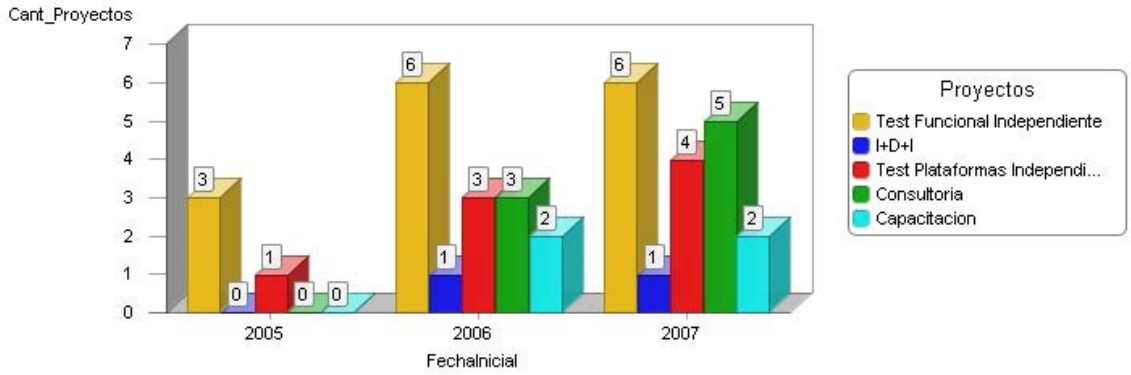

Figura 1. Cantidad de Proyectos por tipo y año

El monto total de ingreso de los distintos proyectos puede clasificarse de la siguiente manera:

- un 89% corresponde a organizaciones privadas y un 11% a organizaciones públicas
- un 33% corresponde a pruebas funcionales independientes, un 33,6% corresponde a consultoría, un 30% a ensayos de plataformas y un 3,4% a capacitación.
- Un 90% son proyectos en Uruguay y un 10% son proyectos en el extranjero
- Un 70% de los proyectos corresponden a organizaciones que producen software, un 28% a organizaciones que consumen software y un 2% a organizaciones consultoras

Los proyectos de consultoría han sido en su gran mayoría para ayudar a formar el área de pruebas en las organizaciones y/o mejorar su metodología de pruebas. La estrategia que se presenta en este artículo fue concebida inicialmente para las pruebas funcionales independientes, donde un equipo del CES es contratado para probar un producto de software dentro de un intervalo de tiempo definido. Dicha estrategia luego también ha sido utilizada en consultorías, donde las organizaciones adoptan esta estrategia para probar sus

propios productos, la estrategia de gestión se adapta muy bien a este contexto. Las áreas de prueba internas de las organizaciones pueden utilizar esta estrategia para planificar sus pruebas funcionales y mantener las actividades de pruebas controladas.

3.1. Metodología para las pruebas utilizada en el CES

ProTest [6] es el proceso para pruebas funcionales de productos de software utilizado en el CES. El proceso tiene cuatro etapas: Estudio Preliminar, Planificación, Ciclo de Prueba y Evaluación del Proyecto, las cuales se muestran en la Figura 2.

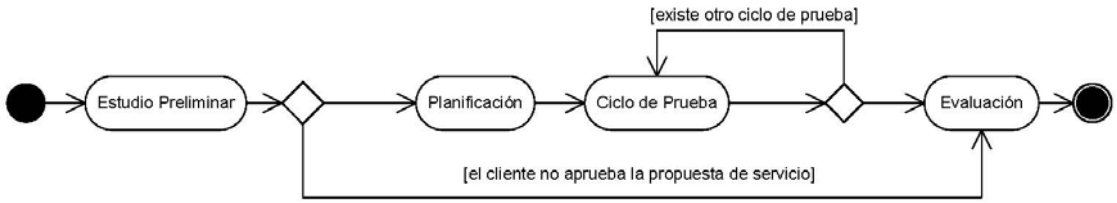

Figura 2. Etapas de ProTest

Los principales objetivos de cada etapa son:

- Estudio Preliminar: En esta etapa, se estudian las principales funcionalidades del producto, con el objetivo de definir el alcance de las pruebas y un primer cronograma de los ciclos de prueba. A partir de estos datos se realiza la propuesta de servicio de prueba. Si el cliente aprueba la propuesta de servicio de prueba, se sigue con la etapa de Planificación, en caso contrario se pasa a la etapa de Evaluación, donde se analiza cuales fueron los problemas en este proyecto y se archiva para su consulta en futuros proyectos.
- Planificación: El objetivo de esta etapa es planificar el proyecto de prueba. Se definen los ciclos de prueba y las funcionalidades a probar en cada ciclo en función del análisis de riesgo del producto. Se genera el Plan de Pruebas que resume toda la información del proyecto de prueba y las decisiones tomadas durante la etapa de Planificación.
- Ciclo de Prueba: El objetivo de esta etapa es generar y ejecutar las pruebas para una versión determinada del producto. El proyecto de prueba es guiado por los ciclos de prueba, cada ciclo de prueba está asociado a una versión del producto a probar.

- Evaluación: Esta etapa tiene como objetivo conocer el grado de satisfacción del cliente, realizar el informe final, evaluar el proceso de prueba para su mejora y almacenar los elementos del proyecto de prueba para su uso en proyectos posteriores.

La etapa Ciclo de Prueba se divide a su vez en cuatro sub-etapas: Seguimiento del Ciclo, Configuración del Entorno, Diseño de las Pruebas y Ejecución de las Pruebas, las cuales se muestran en la Figura 3. A continuación se describen los objetivos de cada sub-etapa dentro del Ciclo de Prueba:

- Seguimiento del Ciclo: El objetivo es realizar el seguimiento y control del ciclo de prueba. La planificación realizada al principio del proyecto es revisada al comenzar cada ciclo de prueba, se planifican en detalle las tareas para el ciclo y se ajusta la planificación según las estimaciones y posteriores mediciones realizadas en los ciclos anteriores.
- Configuración del Entorno: El objetivo es configurar el ambiente de pruebas, separándolo del ambiente de desarrollo, instalar las herramientas necesarias y el software a probar en la versión correspondiente a cada ciclo de prueba.
- Diseño de las Pruebas: Consiste en el diseño de los casos de prueba a partir de la especificación del producto.
- Ejecución de las Pruebas: El objetivo de esta etapa es contrastar el comportamiento esperado del software con su comportamiento real, analizar las diferencias y reportar los resultados.

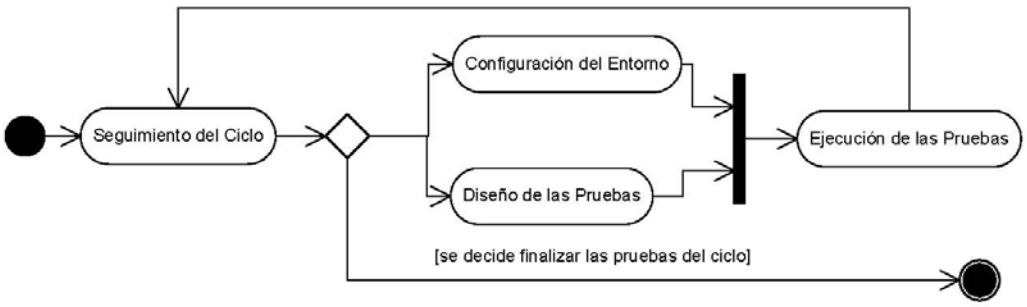

Figura 3. Ciclo de Prueba

Los ciclos de prueba se solapan en el tiempo, mientras se ejecutan las pruebas para un ciclo, se puede al mismo tiempo estar diseñando las pruebas del ciclo siguiente. En cada

ciclo se revisan solamente las especificaciones de las funcionalidades que serán probadas en ese ciclo, con el objetivo de poder diseñar las pruebas que serán ejecutadas en ese ciclo. En [7] y [6] pueden consultarse casos de estudio donde se puso en práctica esta metodología.

4. Cuestiones relativas a la adaptación del proceso software.

En esta sección se presenta la estrategia para la gestión de las pruebas funcionales de un producto de software utilizada en el Centro de Ensayos de Software. Las principales características de la estrategia son que se basa en realizar un estudio de los riesgos del producto que permita definir el alcance para las pruebas, define los ciclos de prueba que se realizarán del producto en función del plan de desarrollo del producto y usa un enfoque iterativo para la planificación de las pruebas, donde se define una planificación general a nivel macro de las funcionalidades a probar en cada ciclo, la que se revisa y refina al comenzar cada ciclo de prueba. Esta metodología fue concebida para realizar pruebas manuales, pero puede ser adaptada para proyectos de automatización de las pruebas, donde la planificación de los casos de prueba a automatizar es realizada siguiendo esta estrategia, como se describe en [8].

4.1. Prueba basada en los riesgos del producto

El enfoque basado en los riesgos tiene tres pasos: primero confeccionar una lista priorizada de riesgos, luego realizar pruebas que exploran cada riesgo, y por último, cuando un riesgo se mitiga y emergen nuevos, se debe ajustar el esfuerzo de la prueba [9]. Un riesgo es la probabilidad de que algo no deseado ocurra. La magnitud de un riesgo es proporcional a la probabilidad y el impacto del problema. A mayor probabilidad de ocurrencia y mayor impacto, mayor es el riesgo asociado a ese problema. Existen dos enfoques heurísticos para el análisis de riesgo, uno "desde adentro hacia fuera" y otro "desde afuera hacia adentro". Son acercamientos complementarios, cada uno con sus fortalezas. El enfoque de adentro hacia fuera pregunta "¿qué riesgos se asocian a esta funcionalidad?", mientras que el enfoque de afuera hacia adentro es el opuesto "¿qué funcionalidades se asocian a esta clase de riesgo?".

El enfoque "desde afuera hacia adentro" estudia una lista de riesgos potenciales y se realiza la correspondencia con los detalles del producto. Con este enfoque, se consulta una lista predefinida de riesgos y se determina si aplican [10].

El enfoque usado en este artículo para el estudio de los riesgos del producto es el denominado "desde adentro hacia fuera", donde se identifican las funcionalidades del producto y los riesgos asociados a cada una, por ejemplo, tendrán mayor riesgo aquellas funcionalidades que en caso de fallar tienen las consecuencias más serias para el negocio o aquellas que tienen mayor frecuencia de uso, ya que si una parte del sistema es usada frecuentemente y tiene un error, su uso frecuente probablemente hará aparecer la falla [9].

4.2. Ciclo de prueba

Durante el ciclo de vida de un producto, sin importar cual sea el proceso de desarrollo, se van generando distintas versiones de la aplicación. Las actividades de la prueba se realizan para una determinada versión del producto, sobre la cual se ejecutan las pruebas y se reportan los incidentes encontrados. Las pruebas que serán ejecutadas sobre una versión son planificadas con anticipación y deberían ser ejecutadas, a menos que las prioridades cambien.

En un ciclo de prueba se puede ejecutar una, alguna o todas las pruebas planificadas para el producto [4]. Uno de los principales desafíos desde el punto de vista de la prueba independiente es estimar cuantos ciclos de prueba se requieren, ya que no todas las versiones que genera desarrollo llegan a ser probadas por el equipo de prueba, entre dos ciclos de prueba podrían existir más de dos versiones del producto generadas por el equipo de desarrollo.

4.3. Casos de estudio: experiencias en organizaciones reales

El alcance de las pruebas indica qué funcionalidades se van a probar y cuales no. Para poder definir el alcance, se divide el sistema en módulos, componentes o subsistemas, no todos los componentes serán probados con la misma importancia y pueden existir componentes que queden fuera del alcance de las pruebas. Cada componente agrupa varias funcionalidades, se dividen las funcionalidades hasta un nivel en el que sea posible definir el alcance. Luego de esto, se analizan las funcionalidades, dando como resultado un Inventario de Pruebas. El inventario es una lista de las funcionalidades del producto de software. Para cada funcionalidad se asigna:

- Identificador: Referencia única a la especificación de requerimientos, donde se encuentra la descripción detallada del mismo.
- Nombre: Descripción breve de la funcionalidad a probar.
- Prioridad: Indica la prioridad que tiene esa funcionalidad para las pruebas. Los valores posibles son: Alta, Media y Baja.

Estos valores se obtienen de utilizar el enfoque basado en los riesgos del producto de la sección 3.1. Se tomará como ejemplo un producto con 10 funcionalidades, se asume que el producto será construido en forma incremental y que se realizarán tres liberaciones internas desde desarrollo al equipo de pruebas antes de ser liberado el producto al ambiente de producción del cliente. En la Tabla 1 se muestra un ejemplo de la priorización realizada para dicho producto.

Inventario de Pruebas		
Identificador	**Nombre**	**Prioridad**
1	Funcionalidad 1	Media
2	Funcionalidad 2	Alta
3	Funcionalidad 3	Baja
4	Funcionalidad 4	Alta
5	Funcionalidad 5	Media
6	Funcionalidad 6	Baja
7	Funcionalidad 7	Alta
8	Funcionalidad 8	Media
9	Funcionalidad 9	Media
10	Funcionalidad 10	Baja

Tabla 1. Inventario de Pruebas

A partir de la estimación del esfuerzo de probar cada funcionalidad, el plan de desarrollo del producto y la fecha prevista para instalar el producto en el ambiente de producción, se define el alcance para las pruebas. El inventario permite recortar en forma ordenada el alcance, ya que es muy probable que las funcionalidades de prioridad baja sean las primeras en quedar fuera del alcance de las pruebas. La definición del alcance de las pruebas se realiza en reuniones donde participan los representantes del cliente, el gerente de proyecto, el líder de desarrollo y el líder de pruebas. Con un primer alcance para las pruebas definido, el próximo paso es la planificación de las pruebas funcionales en el tiempo, como se describe en la próxima sección.

4.4. Planificación de las pruebas funcionales

Para definir el cronograma de las pruebas a lo largo del proyecto, es necesario conocer el plan de desarrollo que contiene el cronograma de las versiones del producto que se

generarán. Se requiere conocer las fechas en que desarrollo tiene planificado liberar las versiones del producto al equipo de prueba y qué funcionalidades incorpora cada nueva versión. En la Tabla se muestra un ejemplo de cómo podría ser un plan de desarrollo del producto cuyo inventario es el de la Tabla 1, donde se tiene previsto construir el producto incrementalmente, liberando tres versiones intermedias al equipo de pruebas.

A partir de esta información el siguiente paso es estimar cuántos ciclos de prueba de la aplicación se van a realizar. Siguiendo con el ejemplo, se podría definir para el producto de la Tabla 1, que se va a realizar un ciclo de prueba por cada versión del producto y que finalmente se realizará un último ciclo de prueba previo a la instalación en producción para asegurarse que los incidentes encontrados en el último ciclo de prueba fueron solucionados, como se muestra en la Figura 4. En este caso, las fechas “Fecha 4” y “Fecha 5” deben ser acordadas con el equipo de desarrollo y el cliente. Se negocia con los desarrolladores el tiempo que necesitan para realizar las correcciones y generar la nueva versión a probar.

Plan de Desarrollo		
Versión	**Fecha**	**Id. De Funcionalidades que incluye el ejecutable**
1	Fecha 1	1,2,3,4
2	Fecha 2	1,2,3,4,5,6,7
3	Fecha 3	1,2,3,4,5,6,7,8,9,10

Tabla 2. Plan de Desarrollo

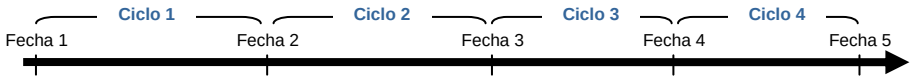

Figura 4. Ciclos de Prueba en el Tiempo

Una vez definidos los ciclos, las siguientes preguntas a responder, son: ¿Qué probar en cada ciclo? ¿Con qué profundidad se realizarán las pruebas de cada ciclo?. A partir de las funcionalidades del Inventario de Prueba de la Tabla 1, se refina cada componente, definiendo las funcionalidades en detalle. Se realiza nuevamente el análisis de riesgo para las nuevas funcionalidades. Puede ocurrir que un grupo de funcionalidades al que se le asignó prioridad alta, al ser dividido en varias funcionalidades, algunas de ellas sean de prioridad alta, otras de prioridad media y otras queden fuera del alcance de las pruebas. En el ejemplo de la Tabla 1, podría ocurrir que la funcionalidad 2, que tiene prioridad “Alta”, al refinarla, resulte en cuatro funcionalidades, como se muestra en la Tabla 2.

Con el inventario refinado, se definen las prioridades para las pruebas de cada ciclo. Del Plan de Desarrollo de la Tabla , surge que para el ciclo 1 sólo se contará con las funcionalidades 1, 2, 3 y 4 del producto. Del inventario de pruebas de la Tabla 2 surge que el orden de prioridad para las pruebas de dichas funcionalidades es: Funcionalidad 2.1, Funcionalidad 2.4 y Funcionalidad 4 con prioridad Alta, Funcionalidad 1 y Funcionalidad 2.3 con prioridades Media y Funcionalidad 1y Funcionalidad 2.2 con prioridad Baja. Ordenado por prioridad las funcionalidades, se obtiene para cada ciclo qué es lo más importante a probar. En la Figura 5 se muestra el orden de las funcionalidades en cada ciclo.

Además de planificar el diseño de las pruebas, se debe planificar el testing exploratorio que se realizará en cada ciclo de prueba. En general, el tiempo con que se cuenta entre un ciclo de prueba y el siguiente no es suficiente como para diseñar todos los casos de prueba de las funcionalidades de ese ciclo. Una posibilidad es complementar el diseño de las pruebas con el *testing* exploratorio. De esta manera, usando el análisis de riesgo del producto, se podría planificar el diseño de las pruebas de las funcionalidades de prioridad alta en cada ciclo y que las funcionalidades de prioridad media y baja, sean exploradas en sesiones de *testing* exploratorio.

Inventario de Pruebas			
Identificador	**Nombre**	**Funcionalidad en Detalle**	**Prioridad**
1	Funcionalidad 1	Funcionalidad 1	Media
2	Funcionalidad 2		Alta
2.1		Funcionalidad 2.1	Alta
2.2		Funcionalidad 2.2	Baja
2.3		Funcionalidad 2.3	Media
2.4		Funcionalidad 2.4	Alta
3	Funcionalidad 3	Funcionalidad 3	Baja
4	Funcionalidad 4	Funcionalidad 4	Alta
5	Funcionalidad 5	Funcionalidad 5	Media
6	Funcionalidad 6	Funcionalidad 6	Baja
7	Funcionalidad 7	Funcionalidad 7	Alta
8	Funcionalidad 8	Funcionalidad 8	Media
9	Funcionalidad 9	Funcionalidad 9	Media
10	Funcionalidad 10	Funcionalidad 10	Baja

Tabla 2. Inventario de Pruebas refinado

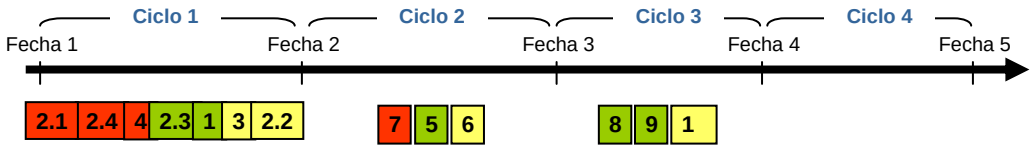

Figura 5. Prioridad de las funcionalidades en cada ciclo

4.5. Planificación de cada ciclo de prueba

Al comienzo del proyecto, se planifican las pruebas a realizar del producto en cada versión que se genere del mismo. Esta planificación debe ser revisada al comenzar cada nuevo ciclo de prueba, ya que los supuestos sobre los que se definió la planificación probablemente hayan cambiado. Por ejemplo, al transcurrir el tiempo pueden cambiar las prioridades de las pruebas debido a cambios en las prioridades del negocio o a la confianza adquirida en el producto como resultado de la realización de las pruebas en ciclos anteriores [4]

También en cada ciclo se deben planificar las pruebas de regresión. Al obtener una nueva versión donde se corrigieron incidentes, se deben ejecutar nuevamente los casos de prueba que encontraron esos incidentes. Como a priori no se conoce cuales ni cuantos serán esos incidentes, al comenzar cada ciclo, deben ser consideradas las pruebas de regresión.

5. Conclusiones

Se ha presentado una estrategia para la gestión de las pruebas funcionales, basada en la importancia de las funcionalidades a probar a través de un análisis de riesgo del producto y combinando los enfoques de técnicas de caja negra y testing exploratorio. Esta estrategia es seguida en el Centro de Ensayos de Software (CES). Aunque fue pensada para un proyecto de prueba independiente, puede ser seguida con éxito como parte del proyecto de desarrollo.

Como líneas de trabajo a futuro en la gestión de las pruebas funcionales, se encuentra lograr mediciones que ayuden en la estimación de las pruebas, según el tipo de producto. Para esto, se requiere tener una base de conocimiento de distintos proyectos de prueba y conocer el esfuerzo requerido para el diseño de los casos de prueba y su ejecución por tipo de funcionalidad a probar. También resulta de interés contar con una estimación del esfuerzo de las pruebas de regresión en cada ciclo.

Referencias

[1] Myers, G. *The art of software testing, 2nd edition*, John Wiley & Sons, 2004.

[2] Beizer, B. *Software testing techniques 2nd edition*, Van Nostrand Reinhold, 1990.

[3] Kaner, C., Falk, J. y Nguyen, H. *Testing Computer Software, 2nd Edition*, John Wiley & Sons, 1999 .

[4] Black, R. *Managing the Testing Process, 2nd Edition*, John Wiley & Sons, 2002.

[5] Bach, J. "Exploratory Testing Explained", *The Test Practicioner*, 2002. Disponible en http://www.satisfice.com/articles/et-article.pdf, accedido: octubre de 2007.

[6] Pérez, B., *Proceso de Testing Funcional Independiente (ProTest)*, Tesis de Maestría en Informática, PEDECIBA Informática, Facultad de Ingeniería, Universidad de la República, Uruguay, 2006.

[7] Pérez, B., Pittier, A., Travieso, M. y Wodzislawski, M., "Testing Exploratorio en la Práctica", *VI Jornadas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC)*, Lima, Perú, pp. 43-49, 2007

[8] Esmite, I., Farias, M, Farias, N, Pérez, B. "Automatización y Gestión de las Pruebas Funcionales utilizando Herramientas Open Source". *XIII Congreso Argentino de Ciencias de la Computación (CACIC 2007)*, Corrientes, Argentina, 2007. Disponible en: http://www.cacic2007.unne.edu.ar/papers/027.pdf, accedido: Noviembre 2007.

[9] Kit E. *Software Testing In The Real World: Improving The Process*, Addison Wesley, 1995.

[10] Bach, J. "Risk-based Testing", *Software Testing and Quality Engineering Magazine* vol. 1, nº 6, noviembre-diciembre, 1999. Disponible en: http://www.stickyminds.com/getfile.asp?ot=XML&id=5009&fn=Smzr1XDD1800filelistfilename1%2Epdf, accedido: noviembre 2007.

Reseña sobre el taller de Pruebas en Ingeniería del Software 2007 (PRIS)

Pablo Javier Tuya-González
Departamento de Informática
Universidad de Oviedo
tuya@uniovi.es

El II Taller sobre Pruebas en Ingeniería del Software (PRIS 2007: http://in2test.lsi.uniovi.es/pris2007/) se celebró en Zaragoza el 11 de Septiembre de 2007, en el marco de las XII Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2007) y del II Congreso Español de Informática (CEDI 2007). Dicho taller fue organizado como parte de las actividades de la Red para la promoción y mejora de las Pruebas en Ingeniería del Software (RePRIS: http://in2test.lsi.uniovi.es/repris/), financiada por el Plan Nacional de I+D+I del Ministerio de Educación y Ciencia y fondos FEDER (acción especial TIN2005-24792-E).

Este taller constituye un foro de discusión las actividades de I+D y de formación en relación con la prueba del software. En esta segunda edición se contó con 22 asistentes, tanto de España como Iberoamérica. Se presentaron un total de 8 contribuciones, seleccionadas tras haber sido sometidas a un proceso de revisión por pares, de las cuales dos fueron procedentes de países de la América hispana y otras dos de empresas del sector.

Las contribuciones cubrieron temáticas realmente variadas. En un primer bloque se presentaron trabajos de investigación. Los dos primeros tratan sobre métodos de pruebas: "Implementación de pruebas del sistema. Un caso práctico" por J. J. Gutiérrez, M. J. Escalona, M. Mejías, A. H. Torres y J. Torres, de la Universidad de Sevilla y "Priorización de casos de prueba mediante mutación" por M. Polo, I. García-Rodríguez y M. Piattini, de la Universidad de Castilla la Mancha. Los dos siguientes son estudios empíricos, uno sobre pruebas en bases de datos: "Un experimento controlado sobre pruebas de consultas SQL" por J. Tuya, M. J. Suárez-Cabal, C. de la Riva, de la Universidad de Oviedo y J. Dolado, de la Universidad del País Vasco y otro sobre pruebas en el entorno industrial "Un experimento sobre hábitos de pruebas artesanales de software: Resultados y Conclusiones" por P. J. Lara-Bercial y L. Fernández-Sanz, de la Universidad Europea de Madrid.

En el segundo bloque se presentaron trabajos relacionados con la experiencia industrial, de los cuales el primero aborda la problemática de la formación en pruebas: "Modelo para la Capacitación de los Especialistas en Pruebas de Sistemas Software" por M. A. García-Palomo y M. Elcuera, de la empresa Métodos y Tecnología (MTP). El segundo muestra la experiencia de la implantación de un método de pruebas: "Aplicación de un Método para Especificar Casos de Prueba de Software en la Administración Pública" por E. Méndez, M. Pérez, L. E. Mendoza, de la Universidad Simón Bolívar. Los dos últimos abordan aspectos generales sobre las pruebas en el entorno industrial: "Gestión de las Pruebas Funcionales" por B. Pérez-Lamancha, del Centro de ensayos de Software de Montevideo y "Casi todas las pruebas del software" por E. Raja-Prado de la empresa NTE S.A.

Finalmente, debemos agradecer su colaboración a los organizadores de las conferencias que albergaron este taller, a los participantes y miembros de la red RePRIS, al MEC como entidad financiadora, y en especial al editor de REICIS por el soporte e interés mostrado en este taller. Todos ellos contribuyen tanto a la consolidación de diversos grupos de investigación en pruebas como a la aportación de una visión práctica que pueda mejorar finalmente la formación y los procesos de pruebas realizados en la práctica industrial.

REICIS

ACTUALIDAD INVITADA

El papel de INTECO en la promoción de la calidad del software como factor clave para el impulso de la industria española

Pablo Pérez San-José
Gerente del Observatorio de la Seguridad de la Información, Instituto Nacional de Tecnologías de la Comunicación (INTECO)
C/ Moisés de León, 57
Edificio Bordadores II
24006 León
pablo.perez@inteco.es

Introducción

En la actualidad, buena parte de los analistas consideran a España como uno de los países mejor posicionados para acoger el modelo *nearshore* de producción de software para Europa; por ello, tanto la competitividad del sector como la certificación reconocida del mismo son claves en la estrategia de impulso de esta industria. Por otro lado, disponer de un software de calidad constituye una ventaja comparativa para todas las organizaciones, ya que contribuye a la disminución de errores y de tiempo invertido, lo que deviene en un aumento de la productividad.

El Plan Avanza contempla un apartado dedicado específicamente a la "mejora de la calidad del software" como una de las Medidas de Política Industrial de Tecnologías de Información y Comunicación. Así, se establecen acciones complementarias que tendrán como objeto la promoción y difusión entre las empresas, en particular las pymes, de modelos contrastados de calidad del software, en línea con el objetivo señalado en el Programa Nacional de Tecnologías Informáticas.

El Instituto Nacional de Tecnologías de la Comunicación (INTECO) –sociedad estatal adscrita al Ministerio de Industria, Turismo y Comercio a través de la Secretaría de Estado

de Telecomunicaciones– ha sido concebido como una plataforma para el desarrollo de la Sociedad de la Información en España, para lo cual gestiona, asesora, impulsa y difunde diferentes proyectos en la estrategia del Gobierno contenida en el Plan Avanza. Así, a través de su Línea de Calidad del Software, INTECO promueve estrategias en materia de calidad en la industria española del software, así como el desarrollo y asentamiento de proyectos empresariales en España; en dicha área de actuación se enmarca el Laboratorio Nacional de Calidad del Software.

De este modo, el LNCS nace con la finalidad de lograr la mejora significativa y mensurable de la calidad del software producido en España y con ello potenciarla como variable estratégica en la competitividad de las organizaciones. Para poder lograrlo, es necesaria una evaluación previa de la situación actual en la que se encuentra industria TIC española.

El diagnóstico de la calidad del software en España

El diagnóstico forma parte del planteamiento metodológico de INTECO, ya que sobre la base del mismo se asienta el diseño de todos los servicios que presta. Por otra parte, las estadísticas e indicadores –necesarios para diseñar, orientar y medir el impacto de las políticas públicas en la materia– no siempre existen y en ocasiones no proceden de fuentes suficientemente rigurosas o independientes. Por esta razón, y como paso previo a la implementación de medidas encaminadas al fomento de la calidad del software, es necesario establecer un análisis y diagnóstico de la situación de partida.

Con este objetivo, durante 2007 el Observatorio de INTECO ha puesto en marcha los tres proyectos de investigación siguientes:

- "La certificación de la calidad como medio para impulsar la industria de desarrollo de software en España", estudio que culmina con una guía de certificación SW para empresas y con unas recomendaciones de actuación enfocadas a las pymes de desarrollo de software.
- "Metodologías y herramientas empleadas en los proyectos de software en España", con el objetivo de identificar necesidades, tendencias y oportunidades del tejido industrial español y con ello perfilar las recomendaciones a las pymes para mejorar los proyectos de software.

- "El Modelo de Factorías de Software con un enfoque nearshore: experiencias de éxito", que pretende analizar el mercado español de Software Factories con el objetivo de posicionar España como destino preferente en la inversión de este tipo de empresas y con ello impulsar el proceso de creación de empresas TIC en regiones con escasa tradición tecnológica.

Las conclusiones de estos estudios permitirán orientar diversas estrategias de apoyo al sector desarrolladas por el LNCS.

El Laboratorio Nacional de Calidad del Software

El Laboratorio Nacional de Calidad del Software persigue un aumento de la competitividad del sector TIC dedicado al desarrollo informático mediante la introducción de metodologías de calidad internacionalmente reconocidas, que mejoren el posicionamiento de ese segmento empresarial –constituido mayoritariamente por pequeñas y medianas empresas– tanto en el ámbito nacional como en el internacional.

Con esta finalidad, actuará como referente tecnológico nacional del sector, enfocando su actividad al fomento de la cultura de mejora de la calidad entre empresas y administraciones públicas y al desarrollo y asentamiento de proyectos empresariales en España.

Entre sus objetivos y actuaciones se encuentran:

- Sensibilizar al sector sobre la necesidad de implantar esquemas de calidad de los desarrollos y en los procesos de gestión empresariales, mediante acciones de difusión que promuevan la cultura de la calidad del software.
- Aportar un valor añadido a los procesos de certificación de calidad de producto introduciendo el consenso en el sector TIC sobre la certificación de productos a partir de acciones de apoyo y estructuración del mercado (consorcios y asociaciones empresariales e institucionales).
- Formar especialistas en la gestión de la calidad, así como mejorar la capacitación metodológica y de aplicaciones de los desarrolladores del software mediante cursos, herramientas y servicios de apoyo.
- Desarrollar servicios públicos destinados a pymes y a usuarios finales de las TIC en materia de buenas prácticas de calidad.

- Asesorar en la realización de planes de mejora para situar a la empresa o administración pública en un estado de madurez que le permita poder afrontar identificar, diseñar e implementar dichos planes.
- Constituir un Centro Demostrador de Soluciones y Tecnologías al servicio de las empresas desarrolladoras, distribuidoras y consumidoras de software. Pretende establecerse como catálogo y escaparate de las mejores prácticas, metodologías y herramientas aplicables a los procesos del software, primando la calidad como elemento diferenciador del mercado español.
- Reforzar la visibilidad de la industria del software española en los mercados internacionales.

Todas estas acciones están dirigidas tanto a las empresas como a las administraciones públicas. No obstante, el segmento de la pequeña y mediana empresa será objetivo prioritario, ya que su acceso a este tipo de herramientas no está lo suficientemente desarrollado debido a restricciones económicas y a la falta de sensibilización en la materia.

Un primer ejemplo de la labor de apoyo y difusión de proyectos relacionados con el desarrollo de aplicaciones informáticas y sistemas de comunicación llevada a cabo por el Laboratorio Nacional es el innovador proyecto de Receta Veterinaria Electrónica. Éste se basa en el DNI electrónico como herramienta de autenticación y firma de las recetas por los veterinarios, así como en una aplicación para la revisión, gestión y dispensación de medicamentos para las empresas farmacéuticas.

Promoción de estándares y normalización

Finalmente, el compromiso de INTECO con la industria informática se evidencia en sus acciones de apoyo a la generación, difusión y consolidación de estándares que aporten confianza y estabilidad al mercado en cuanto a la producción, distribución e identificación de productos y servicios de calidad.

En este sentido INTECO ha liderado un proyecto para impulsar en el ámbito de las administraciones públicas el lenguaje XBRL, estándar ampliamente utilizado en los mercados financieros. En concreto, se ha desarrollado un proyecto de implantación de XBRL en la transmisión de datos en la liquidación presupuestaria de las entidades locales. La experiencia, que se inició a finales de 2006, ha contado con la participación de la

Dirección General de Coordinación Financiera con las Entidades Locales del Ministerio de Economía y Hacienda, la Asociación XBRL España, Caja España y el Ayuntamiento de Cacabelos (municipio leonés pionero en su implantación) y dado como fruto la taxonomía XBRL de intercambio de datos financieros de las entidades locales.

Referencias

Más información en www.inteco.es

Perfil profesional

Pablo Pérez San-José es gerente del Observatorio de INTECO y director técnico de los referidos estudios sobre calidad del software desarrollados por esta entidad. Licenciado en Economía y en Derecho por la Universidad Carlos III de Madrid y D.E.A. en Economía por la UNED, ha desarrollado su carrera profesional en la dirección de Análisis Económico y Mercados y la dirección de Estudios de la Comisión del Mercado de las Telecomunicaciones (CMT), así como en el área de estudios del Observatorio de Red.es, siendo miembro del equipo de trabajo del Grupo de Análisis y Prospectiva del sector de las Telecomunicaciones (Gaptel).