MÉTODO PARA GENERAR CASOS DE PRUEBA FUNCIONAL EN EL DESARROLLO DE *SOFTWARE*

Liliana González Palacio*

Recibido: 31/08/2009
Aceptado: 05/10/2009

RESUMEN

Un aspecto crucial en el control de calidad del desarrollo de *software* son las pruebas y, dentro de estas, las pruebas funcionales, en las cuales se hace una verificación dinámica del comportamiento de un sistema, basada en la observación de un conjunto seleccionado de ejecuciones controladas o casos de prueba.

Para hacer pruebas funcionales se requiere una planificación que consiste en definir los aspectos a chequear y la forma de verificar su correcto funcionamiento, punto en el cual adquieren sentido los casos de prueba. En este artículo derivado de investigación se define un método para generar casos de prueba funcional a partir de casos de uso del sistema, como producto intermedio del proyecto cofinanciado titulado "Herramienta para la documentación de pruebas funcionales".

Palabras clave: pruebas de *software*, casos de prueba, ingeniería de *software*, pruebas funcionales.

* Ingeniera de sistemas Universidad de Antioquia. Magíster en Ingeniería con énfasis en Informática Universidad de Antioquia. Docente tiempo completo Programa Ingeniería de Sistemas Universidad de Medellín. Teléfono: 3405529. E-mail: ligonzalez@udem.edu.co

GENERATING FUNCTIONAL TESTING CASE METHOD IN SOFTWARE DEVELOPMENT

ABSTRACT

Testing is a main aspect in quality control of software development, especially functional tests. The aim of functional testing is to dynamically verify the system behavior, based on the observation of a given set of controlled executions or test cases.

Planning is required to make functional tests, defining the aspects to be checked and the way to verify its proper operation; this allows test cases make sense. In this paper (research based), we propose a method to generate functional test cases from system use cases, based on the co-financed project "Tool for Documenting Functional Testing."

Key words: software testing, test cases, software engineering, functional testing

INTRODUCCIÓN

Un aspecto crucial en el control de calidad del desarrollo de *software* son las pruebas y, dentro de estas, las pruebas funcionales, en las cuales se hace una verificación dinámica del comportamiento de un sistema, basada en la observación de un conjunto seleccionado de ejecuciones controladas o casos de prueba [1].

Las pruebas funcionales son aquellas que se aplican al producto final, y permiten detectar en qué puntos el producto no cumple sus especificaciones, es decir, comprobar su funcionalidad [2]. Para realizarlas se debe hacer una planificación que consiste en definir los aspectos a examinar y la forma de verificar su correcto funcionamiento, punto en el cual adquieren sentido los casos de prueba.

En este artículo se define un método para generar casos de prueba funcionales a partir de casos de uso del sistema, como producto intermedio del proyecto de investigación titulado "Herramienta para la documentación de pruebas funcionales", y está organizado como se indica a continuación: en la segunda sección se encuentran los materiales y métodos que fundamentan el trabajo. La tercera sección presenta los resultados, esto es, el método propuesto en este artículo. La discusión de resultados es mostrada en la sección 4. Las conclusiones y trabajos futuros se enuncian en la quinta sección. Por último las referencias.

1. MATERIALES Y MÉTODOS

En esta sección se presentan algunos conceptos básicos que permiten entender las secciones siguientes. Inicialmente se indica a manera de glosario la terminología necesaria, pasando por una breve revisión de la literatura en cuanto a métodos existentes para derivar casos de prueba y por último la propuesta que ocupa esta publicación.

- Caso de prueba [2]: conjunto de guías que incluye pasos y resultados esperados durante la ejecución de una prueba funcional del *software*.
- Caso feliz: caso de prueba que prueba el funcionamiento del flujo normal del caso de uso relacionado.
- Verificación [3]: Conjunto de actividades que pretenden resolver el interrogante: ¿Se está construyendo el producto correctamente?
- Validación [3]: Conjunto de actividades que pretenden resolver el interrogante: ¿Se está construyendo el producto correcto?
- Error [2]: Discrepancia entre los resultados obtenidos al ejecutar el programa y los resultados que se esperaban.
- Escenario [4]: Conjunto ordenado de interacciones entre un sistema y uno o varios actores.
- Caso de uso [5]: conjunto de escenarios.
- Condiciones de ejecución en un caso de prueba [2]: inventario de datos con los cuales se llevarán a cabo cada paso indicado en el caso de prueba.
- Nivel de complejidad de un error [2]: impacto que genera la presencia del error detectado en caso de no ser resuelto y ser liberada la aplicación sin corregirlo.
- Resultado esperado: reacción ideal (lo que desea el cliente y lo que está en el documento de requisitos) que debe tener la aplicación ante un escenario y condiciones de ejecución indicadas.

Para llegar a la aproximación propuesta se hizo un sondeo de las formas actuales usadas para derivar casos de prueba funcional. A continuación se presenta de manera concisa la revisión de la literatura.

La metodología SCENT [4] permite derivar casos de prueba tomando como insumo la defini-

ción de escenarios y actores que interactúan con el sistema, para luego definir prioridades, pasando por la elaboración de diagramas de dependencia, diagramas de estados y por último generar casos de prueba.

Heumann [6] desarrolla un método para generar casos de prueba tomando como base casos de uso, e identificando dentro de cada uno los posibles escenarios, o caminos de ejecución, y por último definir los valores a probar de cada caso de prueba. Finalmente se obtiene una lista de casos de prueba, con los valores que deben probar y los resultados esperados para cada caso.

La propuesta de Riebisch et al. [7] está centrada en la transformación automática de un modelo de casos de uso a un modelo de uso que sirve como entrada para realizar pruebas estadísticas automáticas, que mejoran el nivel de cobertura, partiendo de que diferentes partes de un *software* no necesitan ser probadas con la misma minuciosidad. El método comienza con el refinamiento de los casos de uso ampliándolos con precondiciones y postcondiciones, alternativas al camino de ejecución principal y referencia a otros casos de uso relacionados. Después se traducen a diagramas de estado y se elabora el modelo de uso donde se indica la probabilidad de que ocurra una transición y se identifican los caminos de ejecución más frecuentes. Por último, se extraen los modelos de prueba a partir de los modelos de uso y se generan recorridos aleatorios sobre cada modelo de uso. Cada camino aleatorio será un caso de prueba.

De otro lado, Hartman [8] es una metodología centrada en dos productos: el primero compuesto por un modelo del sistema escrito en el lenguaje de modelado IF y un conjunto de diagramas UML de clases y estados que van a permitir la generación automática del conjunto de pruebas. El segundo, conformado por un conjunto de objetos de casos de prueba ejecutables tanto en el modelo del sistema como en la implantación, lo que permite comparar los resultados esperados y los obtenidos. Para obtener los productos enunciados, en primer lugar se construye un modelo de comportamiento del sistema a partir de sus especificaciones. Este modelo está compuesto por diagramas UML de clases y un diagrama UML de estados por cada clase que describe el comportamiento de los objetos de dicha clase. A continuación se elaboran los objetivos de las pruebas (pruebas de casos de uso con datos concretos, pruebas de carga del sistema, etc.) y se traducen a un conjunto de directivas de generación y ejecución de pruebas. En el siguiente paso, una herramienta genera automáticamente una serie de pruebas que satisfacen los objetivos de prueba anteriores y se ejecuta automáticamente. Por último, se analizan los resultados y se repiten los pasos hasta que se alcanzan los objetivos deseados.

Las anteriores propuestas, reconociendo que se encuentran muy bien estructuradas, no facilitan la derivación de casos de prueba por la cantidad excesiva de pasos y modelos intermedios que se deben fabricar antes de llegar al resultado final. Este tipo de procedimientos no resulta práctico para una empresa dedicada al oficio de las pruebas. La aproximación propuesta en este artículo no requiere de modelos intermedios y cuenta con una lista de chequeo que permite tener en cuenta aspectos cruciales a la hora de hacer pruebas funcionales.

2. RESULTADOS

En esta sección se presenta el método propuesto para derivar casos de prueba funcional, a partir de un ensamble entre las aproximaciones estudiadas y la experiencia de la empresa con la cual se desarrolla este proyecto cofinanciado.

La figura 1 muestra los insumos y productos que se generan durante el proceso de diseño de casos de prueba del método propuesto:

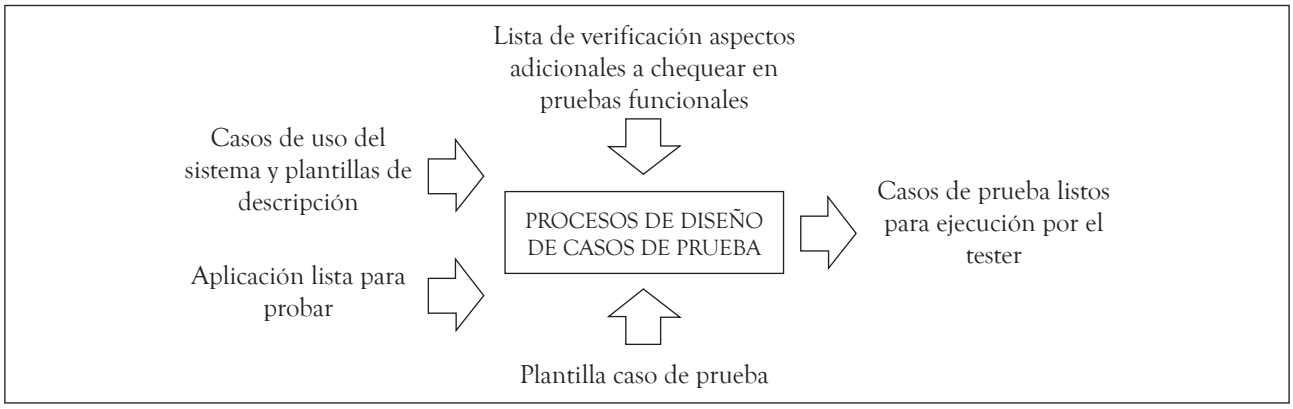

Figura 1. Entradas y salidas del proceso de diseño de casos de prueba

Fuente: Elaboración propia

Como se muestra en la figura 1, para derivar casos de prueba funcional usando el método propuesto en este artículo se debe contar con insumos como: la especificación de casos de uso (diagramas y plantillas de descripción de cada caso de uso), una lista de chequeo que permita determinar si ya fueron probados todos los aspectos relevantes del *software*, una plantilla para diligenciar cada caso de prueba y la versión de la aplicación a probar. Como salida se obtendrá un conjunto de plantillas de casos de prueba debidamente diligenciados.

Luego de contar con estos insumos será sencillo derivar los casos de prueba funcional de una aplicación. A continuación se presenta un diagrama que permite observar su proceso de diseño:

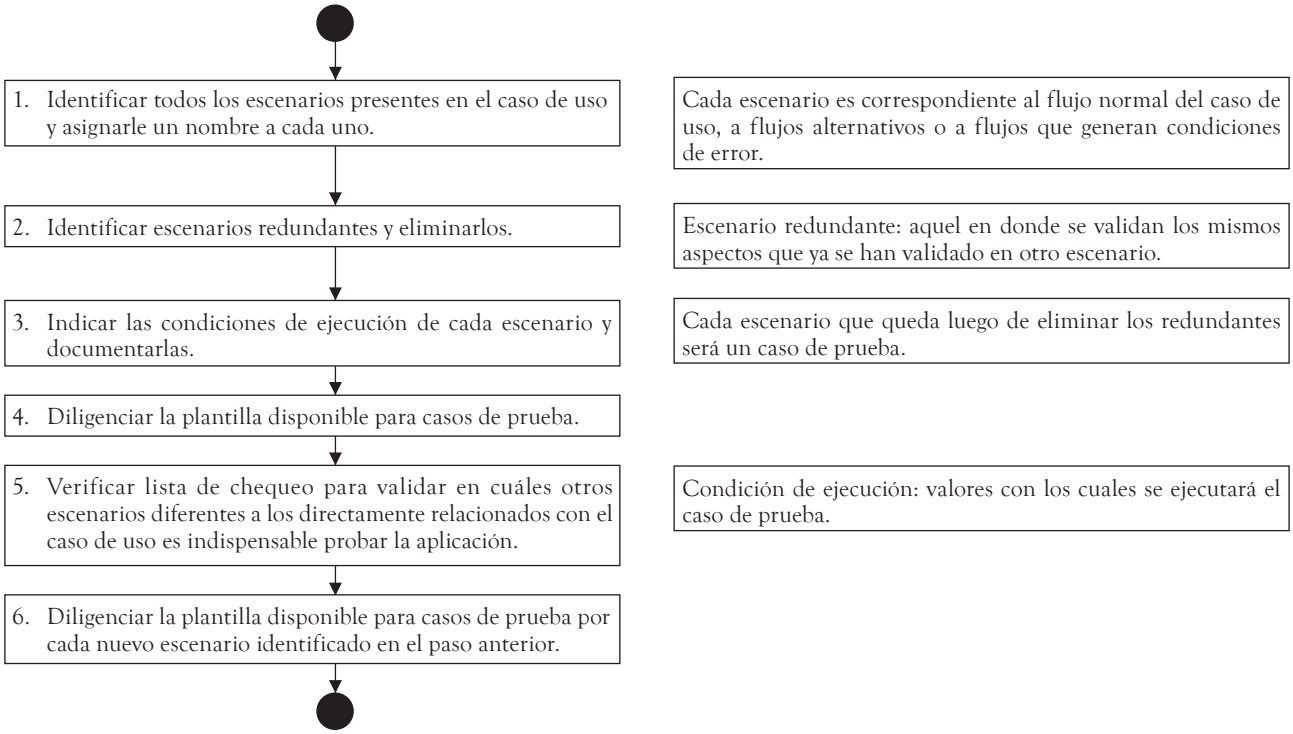

Figura 2. Proceso para derivación de casos de prueba funcional.

Fuente: elaboración propia

La plantilla de caso de prueba que se referencia en los pasos 4 y 6 del diagrama se presenta a continuación:

Tabla 1. Plantilla para caso de prueba funcional

Campo a diligenciar	Orientación para el diligenciamiento
ID caso de prueba	Identificador único para el caso de prueba
Nombre caso de prueba	CasoUso_aspectoaProbar
Descripción	Se probará la respuesta del sistema cuando se presenta X escenario...
Precondiciones	Condiciones que se deben cumplir para la ejecución del caso de prueba.
Relaciones CdeU	Indicar si el caso de uso al cual se le están derivando los casos de prueba tiene conexiones con casos de uso mediante relaciones de include, extend o generalización.
Pasos y condiciones ejecución	Pasos detallados del caso de prueba y los datos con los que se probará
Resultado esperado	Resultado ideal de la aplicación de acuerdo a los pasos ejecutados
Estado caso de prueba	• Ejecutado Exitoso Fallido Frenado • Pendiente de ejecución • En construcción
Resultado obtenido	Se diligencia luego de la ejecución del caso de prueba y de acuerdo a la reacción de la aplicación
Errores asociados	Al ejecutar el caso de prueba, en este campo se indican los identificadores únicos de los errores presentados
Responsable diseño	Nombre del analista de pruebas que diligenció la plantilla
Responsable ejecución	Nombre del tester que ejecutó el caso de prueba
Comentarios	

Fuente: Elaboración propia

La lista de chequeo referenciada en el paso 5 de la figura 2 permite identificar otros aspectos más específicos que se le deben probar a cada funcionalidad de la aplicación:

1. Campos opcionales (resuelven dudas como ¿En el sistema se están respetando los datos que son opcionales? ¿Es posible lograr la funcionalidad si el usuario deja en blanco estos campos opcionales?).
2. Campos obligatorios (permiten detectar si: ¿El sistema está permitiendo que se dejen en blanco campos marcados como obligatorios? ¿Qué pasa si se deja en blanco uno de estos campos?).
3. Tamaño permitido en los campos (logran aclarar comportamientos como: ¿Qué pasa si en un campo que tiene longitud de 10 se ingresan más caracteres?).
4. Tipos de datos permitidos (aclaran aspectos como: ¿Si en un campo que es tipo numérico se ingresan otros caracteres que pasa? ¿El sistema lo controla?).
5. Cálculos (¿Si un campo depende de valores que se ingresan en otros campos, que pasa entonces si se ingresan valores incorrectos?).
6. Formato de los datos (¿Si un campo debe tener formato de pesos, se está respetando esto? O

solo se ponen cifras y no se indican unidades de medida? ¿son peras, manzanas o qué?).

7. Funcionamiento de vínculos.

3. DISCUSIÓN DE RESULTADOS

En esta sección se muestra un comparativo de los métodos estudiados y el propuesto en este artículo tomando como base algunos criterios importantes:

Tabla 2. Tabla comparativa de métodos estudiados

Criterio de comparación/ propuesta revisada	1	2	3	4	5
Cantidad de pasos para generación de casos de prueba	19	3	6	6	6
Modelos intermedios antes de la generación de casos de prueba	Sí	No	Sí	Sí	No
¿Se parte de los requisitos funcionales para la generación de casos de prueba?	Sí	Sí	Sí	Sí	Sí
¿Se cuenta con una lista de chequeo de aspectos a probar?	No	No	No	No	Sí
¿Se cuenta con ejemplos de uso?	Sí	Sí	No	Sí	Sí
Nivel de dificultad (escala de 1 a 5)	5	3	4	4	2

Fuente: Elaboración propia

Los números que encabezan cada columna indican las propuestas revisadas:

1: Metodología SCENT [4]; 2: Método de Heumann [6]; 3: Método de Riebisch [7]; 4: AGEDIS [8]; 5: Método propuesto en este artículo.

Una explicación de cada criterio de comparación se indica a continuación:

Tabla 3. Criterios de comparación de los métodos estudiados

Criterio de comparación/ propuesta revisada	*Descripción del criterio*
Cantidad de pasos para generación de casos de prueba	Se indica el número de pasos que se deben seguir para derivar casos de prueba.
Modelos intermedios antes de la generación de casos de prueba	Pone manifiesto si se requieren modelos intermedios para llegar al conjunto de casos de prueba
¿Se parte de los requisitos funcionales para la generación de casos de prueba?	Se indica si los requisitos están como insumo principal para la construcción de pruebas funcionales.
¿ Se cuenta con una lista de chequeo de aspectos a probar?	Evidencia si se tienen herramientas de apoyo como un listado de aspectos a probar.
¿Se cuenta con ejemplos de uso?	Se refleja la facilidad de entendimiento del método por los ejemplos existentes de su uso.
Nivel de dificultad (escala de 1 a 5)	Cuantifica el nivel de dificultad de uso teniendo en cuenta los criterios anteriores.

Fuente: Elaboración propia

Como es posible observar en la tabla 2, algunas propuestas cuentan con un número excesivo de pasos para su utilización, además de agregar esfuerzo adicional al exigir la generación de modelos o diagramas intermedios antes de la derivación de casos de prueba. Por otro lado, no cuentan con guías como listas de chequeo que permiten validar si se tuvieron en cuenta aspectos críticos en la prueba funcional. Adicional a esto tampoco se encontró en la literatura el conjunto de plantillas usadas para abordar cada paso indicado en el método. Por último, algunas de las propuestas estudiadas tienen un alto nivel de dificultad asociado con los

modelos intermedios y con la falta de plantillas y guías para su uso.

La propuesta objeto de este artículo recoge los métodos existentes y la experiencia empresarial para generar un método con un nivel de formalidad adecuado, y reduce algunas desventajas como el nivel de dificultad y la cantidad de modelos intermedios, evidenciados en las otras aproximaciones.

4. CONCLUSIONES

La planificación y diseño de pruebas funcionales en las primeras fases del desarrollo ayuda a validar los requisitos funcionales. Durante el diseño de la prueba se busca obtener un conjunto amplio de casos de prueba para chequear que toda la información incluida en los casos de uso efectivamente esté implementada en la aplicación bajo prueba. Pensando en la dinámica de las empresas, se debe contar con un método que facilite la derivación de casos de prueba evitando en lo posible la generación de modelos intermedios que aumentan el tiempo a invertir para el diseño de la prueba. En este artículo se presentó un método sencillo para generar casos de prueba funcional a partir de casos de uso del sistema.

En publicaciones posteriores se presentarán casos de estudio desarrollados usando el método propuesto, ya que por extensión del artículo no es posible incluirlos aquí. Como trabajo futuro se plantea la construcción de una herramienta informática que facilite el uso del método propuesto.

REFERENCIAS

[1] J. Gutiérrez, M. J. Escalona, M. Mejías *et al.*, "Analysis of Proposals to Generate System Test Cases From System Requirements," in CAiSE'05 Forum, Porto, Portugal, 2005.

[2] W. Lewis, *Software testing and continuous quality improvement*, 2 ed., Boca Ratón, FL: Gunnasekaran Veerapillai, technical contributor, 2005.

[3] I. Sommerville, *Software Engineering*, 7 ed., Michigan: Pearson/Addison-Wesley, 2005.

[4] J. Ryser, and M. Glinz, "A Practical Approach to Validating and Testing Software Systems Using Scenarios," in Quality Week Europe QWE'99, Bruselas, 1999.

[5] C. Larman, *UML y patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado*, 2 ed., Madrid: Prentice Hall, 2003.

[6] J. Heumann, "Generating Test Cases From Use Cases," *The rational edge*, http://download.boulder.ibm.com/ibmdl/pub/software/dw/rationaledge/jun01/GeneratingTestCasesFromUseCasesJune01.pdf, 2001].

[7] M. Riebisch, I. Philippow, and M. Götze, "UML-Based Statistical Test Case Generation," in Revised Papers from the International Conference NetObjectDays on Objects, Components, Architectures, Services, and Applications for a Networked World, 2003, pp. 394-411.

[8] A. Hartman, *AGEDIS 1999-20218 Final Project Report*, 2004.