Análisis crítico a las propuestas para generar casos de prueba desde los casos de uso para las pruebas funcionales

Critical analysis of proposals to generate test cases from use cases for functional testing

Edgar Serna Montoya[1] & Fernando Arango Isaza[2]
1. Fundación Universitaria Luis Amigó,
2.Universidad Nacional de Colombia sede Medellín
edgar.sernamo@amigo.edu.co; farango@unal.edu.co

Recibido para revisión 9 de mayo de 2010, aceptado 4 de junio de 2010, versión final 28 de junio de 2010

Resumen— **El hecho de formalizar su conocimiento le permite a las disciplinas ingenieriles lograr resultados predecibles. Lamentablemente el conocimiento utilizado en la Ingeniería de Software puede considerarse de un nivel de madurez relativamente bajo; los desarrolladores se guían por la intuición, la moda o lo que dicta el mercado, en lugar de los hechos o declaraciones indiscutibles propias de una disciplina ingenieril. Las propuestas de prueba determinan los diferentes criterios para diseñar los casos de prueba que se utilizan como entradas para examinar un sistema objeto de estudio; lo que significa que diseñar eficaz y eficientemente los casos de prueba es una condición de éxito para las pruebas. El conocimiento que permita seleccionar un método de prueba y un conjunto de casos de prueba, debe surgir de estudios que justifiquen los beneficios y las condiciones de aplicación de los mismos. Este trabajo analiza el nivel de madurez del conocimiento acerca de los métodos de diseño de los casos de prueba generados para la prueba funcional, mediante un análisis crítico de las propuestas desarrolladas en esta temática.**

Palabras Clave— **Casos de prueba, casos de uso, cuerpo del conocimiento, pruebas funcionales, requisitos funcionales, verificación, validación.**

Abstract—**The fact of formalizing their knowledge allows engineering disciplines to achieve predictable results. Regrettably, the knowledge used in Software Engineering can be considered a relatively low level of maturity, developers are guided by intuition, fashion or the dictates of market, rather than facts or undisputed statements proper to an engineering discipline. The test proposals determine the different criteria to design the test cases that are used as imputs to consider a system under study; what means that to design effectively and efficiently the test cases is a condition of success for testing. The knowledge that allows to select a test method and a set of test cases, must arise from studies that justify the benefits and conditions of their application. This paper analyzes the maturity level of knowledge about design methods of test cases for functional testing through an critical analysis of the proposals developed in this subject-matter.**

Keywords— **Body of knowledge, functional requirements, functional testing, test cases, verification, validation, use cases.**

I. INTRODUCCIÓN

La prueba es la última oportunidad en el proceso de desarrollo de software para detectar y corregir sus posibles anomalías a un costo razonable, ya que la forma generalizada de trabajo utilizada por los profesionales en el área es la de ejecutar la prueba sobre el producto "terminado" [1]; mientras que la práctica enseña que la prueba debe ser una actividad que se desarrolla de forma paralela a todo el proceso del ciclo de vida del producto [2]. "E*s mucho más caro corregir los errores que se detectan cuando el sistema se encuentra en operación*" [3], por lo que es de importancia fundamental poder confiar en que el conocimiento aplicado sea lo suficientemente formal como para obtener resultados predecibles en el proceso de las pruebas.

Los métodos de prueba determinan diferentes criterios para diseñar los casos de prueba que se utilizarán como entradas para el sistema en estudio, lo que significa que un efectivo y eficiente diseño de éstos condiciona el éxito de las pruebas [4]. El conocimiento para seleccionar los métodos de prueba debe provenir de estudios que justifiquen los beneficios y condiciones de aplicación de los mismos; sin embargo, "*los estudios formales y prácticos de este tipo no abundan, por lo que es difícil comparar los métodos de prueba -no tienen una sólida base teórica-, y determinar qué variables de los mismos son de interés en estos estudios*" [5].

En vista de la importancia de contar con un conocimiento formal de las pruebas, en este documento se analiza el nivel de madurez del conocimiento en el área y de los casos de prueba generados para las pruebas funcionales. Para tal propósito se hace un análisis crítico de algunas de las más importantes propuestas para generar casos de prueba a partir de los requisitos funcionales de los sistemas. El objetivo principal es recoger el cuerpo de conocimiento acerca de los aspectos que las propuestas tienen en cuenta para diseñar los casos de prueba y su nivel de madurez, de tal manera que esta información pueda ser útil a los desarrolladores para identificar las condiciones de aplicabilidad de los diferentes métodos y los casos de prueba que generan.

Este documento se estructura de la siguiente forma: en la sección 2 se describe el enfoque desde el cual se seleccionaron los diferentes estudios, las pruebas funcionales; en la sección 3 se presenta el análisis crítico de un grupo de propuestas para diseñar casos de prueba desde los caso de uso para las pruebas funcionales, centrado en el método que emplean para diseñarlos; en la sección 4 se detallan las conclusiones extraídas de este trabajo de investigación; y en la 5 se sugieren algunos aspectos que deberían abordarse en futuros estudios, a fin de aumentar el cuerpo de conocimiento acerca de los métodos y las propuestas de diseño de casos de prueba desde los casos de uso para las pruebas funcionales.

II. LAS PRUEBAS FUNCIONALES

El término "prueba de software" comprende un conjunto de métodos, técnicas y conocimiento, cuyo objetivo es determinar la calidad de un sistema software mediante el análisis del resultado de su funcionamiento. Los métodos de prueba proporcionan diferentes criterios para diseñar el conjunto de casos de prueba que se utilizarán para probar el software, y que permiten agrupar los métodos en familias. De esta manera, los métodos que pertenecen a la misma familia son similares en lo que respecta a la información que necesitan para diseñar los casos de prueba -código fuente o especificaciones- o el cómo se aplican los casos de prueba –flujos de control, flujos de datos, errores típicos, etc. El objetivo de este documento no es describir las características de los métodos de prueba o de sus familias, ya que esta información puede ser obtenida de la literatura clásica relacionada, por ejemplo Beizer [6] y Myers [7], [8]; en cambio, se centra en la descripción y el detalle de las propuestas utilizadas para diseñar casos de prueba desde el enfoque de las pruebas funcionales.

La familia de las pruebas funcionales propone un enfoque en el que la especificación del programa se utiliza para diseñar los casos de prueba. El componente a probar se considera como una caja negra –*Black Box*-, cuyo funcionamiento se determina al estudiar las entradas y las salidas asociadas. Del conjunto de posibles entradas del sistema esta familia considera el subconjunto formado por las entradas que hacen que funcione de forma anormal; y la clave para diseñar los casos de prueba consiste en encontrar las entradas que tienen una alta probabilidad de pertenecer a este subconjunto. Para tal propósito el método divide las entradas al sistema en subconjuntos denominados clases de equivalencia, donde cada elemento que la conforma se comporta de manera similar, a fin de que todos los elementos en ella sean las entradas que causen tanto el funcionamiento normal como el anormal del sistema. Los métodos que conforman esta familia difieren entre sí en cuanto a la rigurosidad con la que cubren las clases de datos seleccionados.

III. PROPUESTAS PARA DISEÑAR CASOS DE PRUEBA DESDE LOS CASOS DE USO PARA LAS PRUEBAS FUNCIONALES

Las propuestas analizadas a continuación no son las únicas que se han promulgado para diseñar casos de prueba desde los casos de uso para las pruebas funcionales, la selección para este análisis se hizo de aquellas propuestas que cumplieran con las características de haberse promulgado a partir del año 2000, y que tuvieran como punto de partida los requisitos funcionales del software especificados en casos de uso.

3.1 Automated test case generation from dynamic models *[9]*

La propuesta parte de un caso de uso descrito en lenguaje natural y anotado en una plantilla recomendada [10]; se estructura en dos bloques, en el primero se realiza la traducción de los casos de uso a diagramas de estados, es decir que se traducen desde el lenguaje natural al diagrama; en la propuesta se incluye un resumen de las reglas que deben aplicarse para generar dicho diagrama desde la definición de los casos de uso [11]. En el segundo bloque, a partir de los diagramas de estados del bloque anterior, se realizan las siguientes actividades: 1) se toman las pre y pos-condiciones del diagrama y se traducen a proposiciones conformadas por un identificador para cada una de ellas; 2) dado que pueden existir proposiciones que no dependen de los estados ni transiciones del diagrama, es necesario hacer una extensión a la actividad anterior, se anexa una proposición en la que el conjunto de datos es válido –está definido-, y otra en la que no es válido –no está definido; 3) sobre el conjunto de proposiciones extendidas se generan las operaciones, proceso que consiste en una traducción sistemática utilizando técnicas de inteligencia artificial; 4) se especifican los estados inicial y final del conjunto resultante de proposiciones -conjunto de requisitos, adicciones y sustracciones-; 5) se define con qué criterio se aplicará la cobertura de la prueba mediante un programa de inteligencia artificial, que toma las transiciones del diagrama de estados y desde su estado inicial analiza el posible estado final; 6) mediante la aplicación de un algoritmo se traduce el diagrama de estados al lenguaje STRIPS [12], y se genera el conjunto de

casos de prueba; el algoritmo permite hallar el conjunto de operaciones que desde las pre-condiciones obtienen las pos-condiciones.

El producto final de esta propuesta es un conjunto de transiciones posibles en el diagrama de estados expresadas con operadores, y las proposiciones iniciales y finales del mismo.

La propuesta hace una explicación detallada del proceso de generación de pruebas que ilustra descriptivamente; puede aplicarse con cualquier herramienta que soporte STRIPS y describe claramente cómo establecer la cobertura. No explica cómo se extraen las proposiciones del diagrama de estados; no tiene en cuenta las dependencias de los casos de uso; trata los casos de uso de manera muy aislada; no automatiza la traducción del caso de uso al diagrama de estados; en la medida que se incrementa la complejidad de los datos también se incrementan proposiciones y operaciones; el hecho de expresar las pruebas con proposiciones y operaciones dificulta la implementación, ya que es necesario realizar retrocesos para describirlas.

3.2 Boundary Value Analysis *[13]*

Es una propuesta que selecciona los datos de prueba de aquellos cuyo valor está a lo largo de sus límites, es decir que selecciona los datos de las fronteras superior e inferior del valor a probar. Incluye los valores máximo, mínimo, justo dentro y justo fuera de los límites, los valores característicos y los valores de error. La idea es que si un sistema funciona correctamente para estos valores, funcionará correctamente para todos los valores entre ellos [14]. Tradicionalmente comienza por identificar el incremento de valor más pequeño en una categoría específica de equivalencia; este incremento se llama el valor límite de *épsilon*, y se utiliza para calcular los valores máximos y mínimos en torno a una clase de equivalencia.

Los pasos para utilizar la prueba de valores límite son simples: primero se identifican las clases de equivalencia; luego se identifican los límites de cada clase y posteriormente se diseñan los casos de prueba para cada valor límite mediante la selección de un punto de la frontera, un punto justo debajo y un punto justo por encima [15]. "Debajo" y "encima" son términos relativos que dependen de las unidades de valor de los datos. Se debe tener en cuenta que un punto justo debajo o encima de un límite, puede estar en otra clase de equivalencia, por lo que no hay ninguna razón para repetir la prueba.

Esta propuesta reduce significativamente el número de casos de prueba que se crean y ejecutan; es más apropiada para sistemas en los que gran parte de la toma de datos para valores de entrada está dentro de rangos o en conjuntos; es aplicable en las pruebas de unidad, de integración, de sistema y de aceptación; todo lo que requiere son entradas que puedan ser particionadas, y fronteras que puedan ser identificadas con base en los requisitos funcionales del sistema; existe mucha documentación, ejemplos y casos de éxito que la soportan [16]-[21].

La herramienta de aplicación es demasiado complicada para utilizar y no ofrece una ayuda clara; su dependencia de otras técnicas, como la de clases de equivalencia, reduce su homogeneidad e independencia, ya que el probador debe conocerlas también.

3.3 Test cases from use cases *[22]*

Parte del principio de que las pruebas se deben diseñar desde las primeras etapas del ciclo de vida del producto, y describe cómo utilizar los casos de uso en la generación de los casos de prueba. El caso de uso se define textualmente en lenguaje natural y en una plantilla.

La propuesta consiste en: 1) generar los escenarios de prueba de los casos de uso, donde se identifican todas las combinaciones posibles entre la ruta principal de ejecución y las alternas, y se enuncian en una tabla; 2) identificar el conjunto de casos de prueba -conjunto de entradas, condiciones de ejecución y resultados esperados- para cada uno de los escenarios y condiciones de ejecución; esta información también se enuncia en tablas pero sin notación o formalismo; 3) identificar el conjunto de valores para cada caso de prueba.

Al final del proceso el resultado es una tabla en la que se describen, en lenguaje natural, todos los casos de prueba que permitan verificar que la implantación del caso de uso es correcta.

Aunque no indica un modelo formal para presentar el caso de uso, sí describe los elementos que debe contener; tampoco indica cómo se obtienen los valores de los datos para el tercer paso; es una propuesta sencilla y simple de aplicar, pero le falta detalle y rigor en la descripción; ofrece poca escalabilidad para procesos más complejos; debido a que trata los casos de uso aisladamente, no es posible observar la dependencia entre ellos; el lenguaje natural en el que está expresada no facilita su automatización; el resultado de aplicarla a casos de uso complejos es un elevado número de casos de prueba; aunque parte del principio de diseñar los casos de prueba desde el comienzo del proyecto, no explica cómo hacerlo; y no describe las reglas sistemáticas que permitan aplicar los pasos.

3.4 Requirement Base Testing *[23], [24], [25]*

Propuesta que parte del conjunto de requisitos en lenguaje natural y, mediante la aplicación de dos bloques de actividades, genera los casos de prueba. Como resultado se obtiene un conjunto de casos de prueba expresados en lenguaje natural y estructurado en un modelo causa efecto o estado esperado.

El primer bloque, revisión de requisitos, está dividido en 4 actividades: 1) se analizan los objetivos del sistema y se validan con los requisitos; 2) a esos requisitos se les aplican los casos de uso; 3) se hace una revisión no detallada de ambigüedades, que consiste en eliminar de la descripción de los requisitos todas las palabras y frases ambiguas; 4) los integrantes del

equipo de trabajo, más conocedores del dominio del sistema, revisan la descripción resultante hasta el momento.

El segundo bloque, generación y revisión de casos de prueba, conlleva la realización de 8 actividades: 1) de la descripción de los requisitos se generan los diagramas causa efecto, que luego se traducen a tablas de decisión en las que se incluyen causas, efectos y posibles combinaciones; cada conjunto de combinaciones se constituye en un potencial caso de prueba; 2) se hace una verificación de consistencia del proceso hasta el momento; 3) los ingenieros de requisitos hace una revisión de los casos de prueba generados; 4) los casos de prueba son revisados por los usuarios; 5) los desarrolladores revisan los casos de prueba; 6) sobre el modelo del diseño del sistema se revisan los casos de prueba; 7) las casos de prueba son revisados sobre el código; 8) se ejecutan los casos de prueba.

La propuesta, aunque extensa, no ofrece una documentación adecuada acerca de los productos que se generan en cada actividad, lo que imposibilita su aplicación sin el apoyo de su propia herramienta; el proceso para traducir requisitos al modelo causa efecto no está documentado; no tiene incluida ninguna plantilla, ni propuesta formal de alguna para redactar requisitos; la actividad en la cual se hace la revisión de las ambigüedades se describe muy pobremente y, debido a que se realiza por personas, tiende a no ser verdaderamente objetiva en su resultado; la ausencia de métodos formales imposibilita su automatización; no se encuentran referencias a casos de aplicación ni de éxito; los ejemplos confunden en vez de clarificar; se duplica un proceso ya que el diagrama causa efecto y la tablas contienen la misma información; la documentación de los procesos de aplicación es muy pobre.

Ofrece las ventajas de contar con una herramienta de soporte y de detallar cómo se pueden integrar las actividades de generación de casos de prueba en un proceso de desarrollo.

3.5 Use case derived test cases *[26]*

Esta propuesta parte de un caso de uso descrito en lenguaje natural e indica toda la información que debe contener -nombre, descripción, requisitos, pre y pos-condiciones, flujo de eventos-, y entrega un conjunto de casos de prueba también descritos en lenguaje natural, que define las acciones y verificaciones que realiza en el sistema. El procedimiento consiste en identificar los caminos de ejecución, que son todos los caminos posibles, a partir del flujo de eventos de cada caso de uso; posteriormente cada camino identificado se transforma en un caso de prueba descrito en lenguaje natural.

No se encuentra una ventaja significativa respecto de las demás propuestas; no tiene suficiente documentación; ofrece escalabilidad nula; no describe claramente cómo diferenciar los caminos; no define con claridad el manejo de los requisitos; no es posible seleccionar los casos de prueba –no es aplicable a grandes sistemas-; no tiene una referenciación suficientemente amplia que la soporte; y la descripción en lenguaje natural de los casos de uso imposibilita su automatización.

3.6 Testing from use cases using path analysis technique *[27]*

Esta propuesta parte de un caso de uso que se describe en lenguaje natural, desde el cual se elabora un diagrama de flujo con los posibles caminos que deben ser probados para recorrerlo. Esos caminos deben ser analizados para luego asignarles una puntuación de acuerdo a su importancia y frecuencia; están descritos en lenguaje natural sin adoptar una presentación formalizada; los caminos mejor evaluados se convierten en los casos de prueba a aplicar, representados en un diagrama de flujo. Puede existir más de un caso de prueba probando un mismo camino, ya que es necesario añadirle valores de prueba a cada uno.

Esta propuesta no indica cómo describir los casos de uso, ni utiliza reglas para hacerlo, pero sí indica cuál es la información que debe incluirse al redactarlos; además, el análisis de los atributos no está demarcado y es posible hacerlo con cuántos se desee, tampoco indica cómo realizar la puntuación de los mismos; no detalla cómo aplicar los casos de prueba, ni cuál será el posible resultado que se alcance o su estructura.

El proceso que describe la propuesta es sencillo y relativamente fácil de implementar; la forma de cómo descartar caminos que no aportan a la prueba está bien descrita; es posible probar el comportamiento de cada caso de uso seleccionado de forma muy completa, e incluye un ejemplo paso a paso de aplicación. Pero, no referencia proyectos reales en los que se haya aplicado; el uso de un lenguaje natural para describir casi todo el proceso da pie para ambigüedades y se convierte en un problema al momento de automatizar las pruebas; si un caso de uso es dependiente de otro, no está documentado como probarlo; aunque es una propuesta sencilla, es difícil estructurarla para aplicar en sistemas complejos.

3.7 Requirements by contract *[28]*

La propuesta parte de un diagrama de casos de uso en notación UML y propone cuatro criterios por medio de los cuales es posible recorrer el modelo para obtener los casos de prueba. Se encuentra dividida en dos momentos, en el primero se hace una extensión de los casos de uso UML mediante un lenguaje de contratos que incluye pre y poscondiciones –expresadas mediante proposiciones- y sus parámetros, que en conjunto permiten expresar las dependencias existentes entre los casos de uso; en el segundo momento se detalla la generación automática de los casos de prueba desde la extensión de los casos de uso. Como resultado se obtiene un modelo de casos de uso extendido con contratos -expresados como caminos que recorren la ejecución de las secuencias de casos de uso que satisfacen las pre y pos condiciones-, y un conjunto de casos de prueba para verificar la implementación del modelo

que genera. Este modelo se expresa en un diagrama que refleja el comportamiento del sistema según los casos de uso diseñados. El estado del sistema se representa por cada nodo del modelo –determinado por las proposiciones-, y las instancias se representan por cada transición.

No detalla cómo generar los casos de prueba desde las instancias por medio de las herramientas de generación de pruebas; además, no es posible desarrollar pruebas que verifiquen aisladamente el comportamiento de cada caso de uso; los casos de uso se extienden sin respetar el estándar UML; no hay forma de saber el número de parámetros que debe utilizar cada caso de uso, y no se encuentra una referencia de cómo implementar las pruebas.

Su fortaleza se refleja en la diversidad de criterios de cobertura y su herramienta experimental de soporte –de libre descarga-; las pruebas se pueden generar desde la secuencia de casos de uso; los contratos son muy flexibles para expresar dependencias entre los casos de uso. El mismo equipo de investigación publicó un trabajo en el que describen cómo aplicarla en una familia de productos de software [29].

3.8 Category partition method *[30], [31]*

Describe cómo generar los casos de prueba partiendo de los casos de uso de una familia de productos [32], y extiende la notación de esos casos de uso mediante una plantilla en lenguaje natural [10], que contiene las características comunes a los productos de la familia, lo mismo que los puntos en que cada producto varía de los demás. Este proceso es una adaptación del propuesto por Ostrand y Balcer [33] y actualizado para trabajar con especificaciones de familia de productos que se pueden modelar mediante casos de uso.

El proceso comienza con la representación de los casos de uso del sistema; se determinan los requisitos funcionales y los rangos de datos que cada uno podría tomar –lo realizan los encargados de las pruebas con base en el conocimiento del sistema y su experiencia-; se determinan las restricciones en cada uno de los rangos que generen errores y que reduzcan el número de casos de prueba; se redactan las especificaciones de la prueba, que es un documento en el que se describe la plantilla con la información recolectada en los pasos anteriores; se genera el contexto de la prueba, que consiste en una combinación de los valores encontrados en los rangos de datos; se traduce esa combinación de valores a un lenguaje ejecutable y se reúnen aleatoriamente para ejecutarse en el sistema.

El resultado final de la propuesta es un conjunto de casos de prueba específico para cada producto y el conjunto de casos de prueba común para los productos de la familia. Estos casos de prueba se describen de forma abstracta, por lo que deben refinarse para obtener casos de prueba posibles de ejecutar en el sistema.

Originalmente fue una propuesta de los años 80 diseñada para familias de productos, pero su enfoque y madurez permite considerarla para aplicación en los sistemas de hoy; además, puede diseñarse para aplicar en sistemas que no pertenecen a una familia específica. Aunque es una propuesta mediante la cual es posible generar conjuntos de valores para las pruebas, muchos de los pasos presentan ambigüedad en su definición, ya que quedan supeditados a la experiencia y conocimiento del sistema por parte del equipo de pruebas, por lo que la información acerca de la cobertura de las pruebas tampoco se ofrece adecuadamente; no es posible su automatización total, y no referencia una herramienta que la soporte.

La propuesta describe un ejemplo práctico de aplicación, es posible verificar el comportamiento de los casos de uso, así como su dependencia con otros casos de uso; se puede aplicar desde comienzos del proceso del proyecto, y también reduce el número de casos de prueba generados a través de las restricciones que tiene en cuenta.

3.9 Requirements to testing in a natural way *[34]*

Según sus autores, es un analizador de requisitos que puede ser utilizado para generar pruebas de caja blanca y caja negra; está conformada por 6 actividades dividas en dos bloques El primer bloque lo conforman 5 actividades: 1) se redactan los requisitos en lenguaje natural y en párrafos estructurados; 2) se hace una identificación de cada frase en los párrafos de descripción de los casos de uso; 3) desde cada una de las frases se genera un árbol sintáctico con sus respectivas anotaciones; 4) a partir de cada árbol se obtiene la estructura de representación del discurso [35], en la que se identifica la semántica de cada elemento del árbol; 5) se refina esta estructura y se eliminan las ambigüedades existentes, luego se traduce automáticamente al lenguaje MONA [36], con lo que se genera una máquina de estados finitos.

En el segundo bloque se recorre la máquina de estados finitos para generar los casos de prueba. Se ofrece la posibilidad de recorrer la máquina de distintas formas, por lo que es posible obtener varios conjuntos de casos de prueba que generan explosión de los mismos. Para evitar esto la propuesta propone preguntar al usuario por los requisitos críticos, y luego reducir los recorridos a ellos.

Aunque es una propuesta bien documentada, no incluye ejemplos de las pruebas generadas, ni cómo se recorren y obtienen las pruebas a partir de la máquina de estados; al construir el árbol sintáctico no es posible clarificar entre verbos, sujetos y otros componentes de las frases que describen los casos de uso; no explica si es necesario recorrer mediante pruebas todas las frases de la descripción del caso de uso.

Su principal ventaja es ser de las pocas que detalla un método para procesar requisitos descritos en lenguaje natural, lo que permite automatizar la generación de las pruebas; aunque no se encontró un caso práctico de aplicación, existen herramientas libres que permiten automatizar parte de las actividades y, para

las otras, existen herramientas desarrolladas por los mismos autores, pero a las que no es posible acceder, por lo que no fue posible verificar su aplicación práctica

3.10 A model-based approach to improve system testing of interactive applications *[37]*

Esta propuesta tiene su origen en las investigaciones de la empresa Siemens, las cuales recopila y amplía Ruder [38]. Parte de la documentación en lenguaje formal de los casos de uso, y como resultado de su aplicación se obtiene el conjunto de pruebas para la interfaz gráfica del sistema.

Los pasos son los siguientes: 1) se modela el comportamiento del sistema mediante un diagrama de actividades UML que describe el comportamiento interno de los casos de uso, y especifica los requisitos de prueba –conjunto de estereotipos que permiten interpretar cada actividad-; estos estereotipos indican la pertenencia de las actividades del usuario, del sistema u otro diagrama de actividades, y se interpretan para construir pruebas ejecutables por el generador de pruebas; 2) se diseñan los casos de prueba mediante *scripts* de prueba, que son la descripción de los diagramas de actividades en lenguaje TSL –Test Specification Language- [39]; luego, mediante TDE -Test Development Environment- [38] se traduce de TSL a guiones de prueba, con lo que obtiene un conjunto de *scripts* que pueden ejecutarse en una herramienta de verificación de interfaces gráficas; 3) luego de construir el conjunto de guiones, es posible ejecutarlos sobre el sistema que se está probando, proceso que refina y mejora los mismos *scripts* a medida que se ejecutan.

Esta propuesta es de las pocas que describe cómo generar pruebas ejecutables; utiliza lenguaje formal que, unido al uso de *scripts* mediante estereotipos escalables, facilita su automatización. Sus inconvenientes se reflejan en que se centra en la interfaz gráfica, imposibilitando su aplicación en otras interfaces; el proceso de cómo se obtienen los diagramas de actividades desde los casos de uso no se detalla lo suficiente, lo mismo que cómo se asigna los estereotipos a cada actividad; no especifica si el proceso de pasar los diagrama de actividades a TSL debe hacerse manual o si es automatizable; no es claro si se requiere un solo diagrama de actividades para todo el proceso o es necesario diseñar uno por cada caso de uso; los diagramas de actividades de cada caso de uso aparecen independientes y no se detalla qué hacer con las relaciones entre ellos; no se encuentra una herramienta que permita aplicar la propuesta en un ambiente de laboratorio o de evaluación.

IV. CONCLUSIONES

- Probar un sistema desde la óptica de las pruebas funcionales es verificar que se han implementado bien los requisitos de la especificación funcional, por lo que éstos deben constituirse en la base sobre la que se diseñan los casos de prueba del sistema.
- Diseñar el modelo del sistema teniendo como base su especificación funcional es una tarea en la que se construye un modelo formal -o semiformal-, que debe expresar lo que se espera del comportamiento del sistema de acuerdo con los datos recogidos en los requisitos. Dado que la base sobre la que se diseña son los requisitos expresados en lenguaje natural, el proceso no puede realizarse automáticamente pero sí sistemáticamente, mediante el seguimiento de pasos y fases definidos con claridad.
- Luego del diseño, en la aplicación de la cobertura de la prueba, es necesario seleccionar un criterio mediante el cual se puedan generar los casos de prueba que posteriormente se aplicarán al diseño del modelo. En este análisis el criterio que tiene más acogida entre las propuestas es el de todos los posibles caminos de ejecución; aunque existen otros criterios que también son viables y, que al igual que los analizados, pueden realizarse automáticamente utilizando una herramienta software.
- Otra característica de las propuestas analizadas es que los valores de los datos de entrada al sistema hacen parte del proceso de prueba, por lo que cualquier propuesta debe tener en cuenta cómo generarlos. Para este caso ese proceso debe describirse más detalladamente, ya que la descripción es muy pobre en la mayoría de las propuestas.
- Los resultados esperados son un elemento imprescindible en la descripción de las propuestas de este análisis, y consiste en poder determinar cuál será la respuesta del sistema a la ejecución del conjunto de casos de prueba seleccionado; estos resultados se obtienen luego de aplicar automáticamente las pruebas a una simulación del modelo del sistema. En este análisis, un porcentaje muy pequeño se ocupa de este tema y parece no serles de utilidad para diseñar el conjunto de casos de prueba.
- El paso final en los proceso de prueba analizados es el de ejecutar los casos de prueba sobre el sistema objeto, pero ninguna de las propuestas ofrece el detalle de cómo generarlo mediante un formalismo que pueda traducirse fácilmente a código, por el contrario, la mayoría describen cómo traducirlos a lenguaje natural.

V. ASPECTOS PARA TENER EN CUENTA EN TRABAJOS FUTUROS

El análisis efectuado a las propuestas para diseñar casos de prueba a partir de la especificación funcional, sustenta la falta de un trabajo más profundo en procura de hallar una propuesta con más integración, y que debe desarrollarse a partir de lo siguiente:

- Los puntos comunes de las propuestas que se analizan en este documento, como son: obtener un conjunto de casos de prueba que de alguna manera garantice que el sistema cumple con las especificaciones funcionales; partir de los requisitos funcionales para generar los casos de prueba; utilizar el análisis de caminos o estados posibles; tener en cuenta que los requisitos funcionales necesariamente no cumplen requisitos formales –lenguaje natural para comenzar-; generar el conjunto de casos de prueba de manera automática y sistemática a partir de los requisitos funcionales, mediante alguna herramienta software, y validar los requisitos funcionales desde las primeras fases del desarrollo. Estos puntos deben ser la base desde la que se puedan corregir las falencias en las encontradas y para potenciar sus fortalezas en el nuevo proyecto.
- En lo que respecta al modelo de comportamiento del sistema se debe identificar claramente cuál es la información que, contenida en la especificación funcional, permita generarlo y luego obtener los casos de prueba.
- Debe diseñarse y describirse una plantilla en la que se estandarice la descripción de los requisitos funcionales en lenguaje natural, para lo que se puede pensar en utilizar una pre-existente como la que describe [10] o en desarrollar otra.
- Dado que los casos de uso no son fijos, debe detallarse cuál es el grado de refinamiento necesario para generar los casos de prueba; mirar por ejemplo el refinamiento propuesto por Dustin et al [40].
- Para obtener una propuesta más robusta es necesario incluir, además de los funcionales, otro tipo de requisitos como los de almacenamiento; ya que los valores de salida son tan importantes en la realización de la prueba, con estos requisitos es posible darle mayor cobertura y eficacia a los casos de prueba.
- En las propuestas analizadas se trabaja con un único modelo del sistema en la generación de los casos de uso, lo que crea dificultades para analizar las interacciones entre éstos; poder contar con modelos alternos o sub-modelos basados en un diagrama de estados, facilita el análisis del comportamiento del sistema desde la especificación funcional.
- Otro asunto importante es el relacionado con el criterio de cobertura, y en especial el de cómo utilizar la prioridad de los casos de uso al momento de generar interacciones o secuencias; pueden tenerse en cuenta algunas propuestas desde la ingeniería de requisitos, como la de Robinson [41], Riebisch et al [42] y la de Escalona [43].
- En lo que respecta a los valores de prueba, debe diseñarse un conjunto de reglas concreto y sistemático que permita reducir el número de decisiones que toman los probadores, a lo cual ayudará lo ya expuesto de incluir el proceso los requisitos de almacenamiento.
- El número de pruebas a ejecutar por cada escenario posible de aplicación es una cuestión fundamental cuando se hace análisis de caminos, para solucionarlo es conveniente pensar en un conjunto concreto de valores y generar un escenario de prueba por cada combinación posible.
- Es necesario recurrir a los lenguajes formales para expresar los escenarios de prueba, ya que con éstos es posible generar automáticamente los casos de prueba; para esta tarea es conveniente revisar trabajos como los de Clarke and Wing [44], Lamsweerde [45], Juristo et al [46], Clermont and Parnas [47], Beckert et al [48], Dasso and Funes [49] y Kaner [50]. Esta forma de trabajo permitirá cuantificar el grado de cobertura de la prueba de forma automatizada, lo que aleja la toma de decisiones del actor humano.
- Validar el conjunto de casos de prueba generado debe ser una meta en la nueva propuesta, por ejemplo mediante la cobertura de los requisitos, si la cobertura es la deseada el conjunto es válido.
- Es necesario contemplar la posibilidad de automatizar cada una de las actividades en el proceso de generación del conjunto de casos de prueba; esta meta debe considerarse prioritariamente ya que es una falencia que resta credibilidad y eficacia a las analizadas. Igualmente importante es considerar la posibilidad de generar una herramienta de soporte a la propuesta, de tal manera que sea posible aplicarla a trabajos reales y en situaciones reales.

REFERENCIAS

[1] Kamde, P. M., Nandavadekar, V. D. and Pawar, R. G., 2006. Value of test cases in software testing. Management of Innovation and Technology, Vol. 2, pp. 668-672.

[2] Leon, D., Masri, W. and Podgurski A., 2007. An empirical evaluation of test case filtering techniques based on exercising complex information flows. IEEE Transactions on Software Engineering, Vol. 33, No. 7, pp. 454-477.

[3] Lewis, W. E., 2000. Software testing and continuous quality improvement. Florida: CRC Press. 657 P.

[4] Sinha, P. and Suri N. 1999. Identification of test cases using a formal approach. Proceedings of the Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing. Madison, Wisconsin, USA, pp. 314-321.

[5] Pressman, R., 2005. Software engineering: a practitioner's approach. New York: McGraw Hill. 958 P.

[6] Beizer, B., 1990. Software testing techniques. New York: International Thomson Computer Press. 470 P.

[7] Myers, G. J., 1979. The art of software testing. New York: Wiley-interscience. 255 P.

[8] Myers, G. J., 2003. Principles of functional verification. New York: Newnes. 217 P.

[9] Fröhlich, P. and Link, J., 2000. Automated test case generation from dynamic models. Lecture Notes in Computer Science, Vol. 1850, pp. 472-491.

[10] Cockburn, A., 2000. Writing Effective use cases. New York: Addison-Wesley. 249 P.

[11] Fröhlich, P. and Link, P., 1999. Modeling Dynamic Behaviour Based on Use Cases. 3rd International Software Quality Week Europe QWE. Brussels, Belgium, Paper 9A.

[12] Fikes, R. E. and Nilsson, N. J., 1971. STRIPS: a new approach to the application of theorem proving to problem solving. Artificial Intelligence, Vol. 2, No. 3-4, pp. 189-208.

[13] Hugger, J., 2001. Wellposedness of the boundary value formulation of a fixed strike Asian option. Journal of Computational and Applied Mathematics, Vol. 185, No. 2, pp. 460-481.

[14] Copeland, L., 2004. A practitioner's guide to software test design. Londres: Artech House. 294 P.

[15] Myers, G. J., 2004. The art of software testing. New York: John Wiley & Sons. 255 P.

[16] Schroeder, P. J. and Korel, B., 2000. Black-box test reduction using input-output analysis. the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis. New York, NY, USA, pp. 173-177.

[17] McGee, P. and Kaner, C., 2004. Experiments with high volume test automation. SIGSOFT Software Engineering Notes, Vol. 29, No. 5, pp. 1-3.

[18] Hierons, R. M., 2006. Avoiding coincidental correctness in boundary value analysis. ACM Transactions on Software Engineering and Methodology, Vol. 15, No. 3, pp. 227-241.

[19] Krishnan, R., Krishna S. M. and Nandhan, P. S., 2007. Combinatorial testing: learnings from our experience. SIGSOFT Software Engineering Notes, Vol. 32, No. 3, pp. 1-8.

[20] Tuya, J., Dolado, J., Suarez-Cabal, M. J. and de la Riva, C., 2008. A controlled experiment on white-box database testing. SIGSOFT Software Engineering Notes, Vol. 33, No. 1, pp. 1-6.

[21] Masri, W. Abou-Assi, R. El-Ghali, M. and Al-Fatairi, N., 2009. An empirical study of the factors that reduce the effectiveness of coverage-based fault localization. 2nd international Workshop on Defects in Large Software Systems: Held in Conjunction with the ACM SIGSOFT international Symposium on Software Testing and Analysis, New York, NY, USA, pp. 1-5.

[22] Heumann, J., 2002. Generating test cases from use cases. Journal of Software Testing Professionals The Rational Edge, September Issue, pp. 3-14.

[23] Mogyorodi, G. E., 2001. Requirements-based testing: an overview. 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems. Santa Barbara CA, USA, pp. 286-295.

[24] Mogyorodi, G. E. 2002. Requirements-based testing: ambiguity reviews. Journal of Software Testing Professionals, December Issue, pp. 21-24. 2002.

[25] Mogyorodi, G. E., 2003. What is requirements-based testing? Crosstalk: The Journal of Defense Software Engineering, Vol. 16, No. 3, pp. 12-15.

[26] Wood, D. and Reis, J., 2002. Use case derived test cases. Software Quality Engineering for Software Testing Analysis and Review, Memories, pp. 235-244.

[27] Naresh, A., 2002. Testing from use cases using path analysis technique. International Conference On Software Testing Analysis & Review. Washington D. C., USA, pp. 237-252.

[28] Nebut, C., Fleurey, F., Le, T. Y. and Jézéquel, J-M., 2003. Requirements by contract allow automated system testing. 14th International symposium of Software Reliability Engineering, Memories, pp. 121-131.

[29] Nebut, C. Fleurey, F., Le, T. Y. and Jézéquel, J-M., 2004. A requirement-based approach to test product families. Frank van der Linden, Vol. 30, No. 14, pp. 198-210.

[30] Bertolino, A. and Gnesi, S., 2003. Use Case-based testing of product lines. ACM SIGSOFT Software Engineering Notes, Vol. 28, No. 5, pp. 355-358.

[31] Bertolino, A. and Gnesi, S., 2004. PLUTO: A test methodology for product families. Lecture Notes in Computer Science, Vol. 30, No. 14, pp. 181-197.

[32] Bertolino, A., Fantechi, A., Gnesi, S., Lami, G. and Maccari, A., 2002. Use case description of requirements for product lines. REPL'02, Essen, Germany, Avaya Labs Tech, pp. 12-18.

[33] Ostrand, T. J. and Balcer, M. J., 1988. The category-partition method for specifying and generating functional tests. Comunications of the ACM, Vol. 31, No. 6, pp. 676-686.

[34] Boddu, R. Guo, L., Mukhopadhyay, S. and Cukic, B., 2004. RETNA: from requirements to testing in a natural way. 12th IEEE International Requirements Engineering Conference. Kyoto, Japan, pp. 262-271.

[35] Blackburn, P. and Bos, J., 1994. Working with discourse representation theory: an advance course in computational semantics. California: Stanford CSLI Publications. 254 P.

[36] Henriksen, J. G., Jensen, J., Jrgensen, M., Klarlund, N., Paige, R., Rauhe, T. and Sandholm, A., 1995. MONA: Monadic second-order login in practice. Tools and Algorithms for the Construction and Analysis of Systems, Warsaw, Poland, pp. 10-19.

[37] Hartmann, J., Vieira, M., Foster, H. and Ruder, A., 2004. UML-based test generation and execution. Proceedings of Workshop on Software Test, Analyses and Verification. Banff, AB, Canada, pp. 234-241.

[38] Ruder, A., 2004. UML-based test generation and execution. Siemens Corporate Research Inc. Berlin, Germany, white paper.

[39] Balcer, M., Hasling, W. and Ostrand, T., 1990. Automatic generation of test scripts from formal test specifications. ACM SIGSOFT Software Engineering Notes, Vol. 14, No. 8, pp. 210-218.

[40] Dustin, E., Rashka, J. and McDiarmid, D., 2002. Quality Web systems. New York: Addison-Wesley. 352 p.

[41] Robinson, H., 2000. Intelligent test automation. Software Testing & Quality Engineering, Vol. 2, No. 5, pp. 24-32.

[42] Riebisch, M., Philippow, I. and Ilmenau, M. G. 2003. UML based statistical test case generation. Lecture Notes in Computer Science, Vol. 2591, pp. 394-411.

[43] Escalona, M. J. ,2004. Modelos y técnicas para la especificación y el análisis de la navegación en sistemas software. Tésis PhD. Departamento de computación, lenguajes y sistemas. Universidad de Sevilla. España.

[44] Clarke, E. M. and Wing, J. M., 1996. Formal methods: state of the art and future directions. ACM Computing Surveys, Vol. 28, No. 4, pp. 626–643.

[45] van Lamsweerde, A., 2000. Formal specification: a roadmap. Conference on The Future of Software Engineering. Limerick, Ireland, pp. 147-159.

[46] Juriso, N., Moreno, A. and Vegas, S., 2004. Reviewing 25 years of testing technique experiments. Empirical Software Engineering, Vol. 9, No. 1-2, pp. 7-44.

[47] Clermont, M. and Parnas, D., 2005. Using information about functions in selecting test cases. 1st International Workshop on Advances in Model-Based Testing. St. Louis, Missouri, USA, pp. 1-7.

[48] Beckert, B., Hoare, T., Hahnle, R., Smith, D. R., Green, C.,

Ranise, S., Tinelli, C., Ball, T. and Rajamani, S. K., 2006. Intelligent systems and formal methods in software engineering. IEEE Intelligent Systems, Vol. 21, No. 6, pp. 71-81.

[49] Dasso, A. and Funes, A., 2007. Verification, validation and testing in software engineering. New York: Idea Group Publishing. 443 P.

[50] Kaner, C., 2003. What Is a Good Test Case? STAR East 2003. Orlando, FL, USA, pp. 76-92.

Universidad Nacional de Colombia Sede Medellín
Facultad de Minas

Escuela de Ingeniería de Sistemas

Grupos de Investigación

Grupo de Investigación en Sistemas e Informática

Categoría A de Excelencia Colciencias
2004 - 2006 y 2000.

GIDIA: Grupo de Investigación y Desarrollo en Inteligencia Artificial

Categoría A de Excelencia Colciencias
2006 – 2009.

Grupo de Ingeniería de Software

Categoría C Colciencias 2006.

Grupo de Finanzas Computacionales

Categoría C Colciencias 2006.

Centro de Excelencia en Complejidad

Colciencias 2006

Escuela de Ingeniería de Sistemas
Dirección Postal:
Carrera 80 No. 65 - 223 Bloque M8A
Facultad de Minas. Medellín - Colombia
Tel: (574) 4255350 Fax: (574) 4255365
Email: esistema@unalmed.edu.co
http://pisis.unalmed.edu.co/