Revista de Procesos y Métricas de las Tecnologías de la Información (RPM) ISSN: 1698-2029
VOL. 1, Nº 3, Diciembre 2004, 11-20
Asociación Española de Sistemas Informáticos (AEMES)

MEJORA DE LA CALIDAD EN DESARROLLOS ORIENTADOS A OBJETOS UTILIZANDO ESPECIFICACIONES UML PARA LA OBTENCIÓN Y PRECEDENCIA DE CASOS DE PRUEBA

Luis Fernández Sanz, Pedro J. Lara Bercial
Dpto. Programación e Ingeniería del Software. Universidad Europea de Madrid.
c/Tajo s/n. Villaviciosa de Odón, Madrid, Spain
E-Mail : lufern@uem.es

Juan J. Cuadrado-Gallego
Departamento de Ciencias de la Computación, Universidad de Alcalá
Ctra. de Barcelona, 33.6, Alcalá de Henares, Spain
E-Mail: jjcg@uah.es

Abstract: Simple test cases generation has been one of the main hopes of software developers through the compueters history. But, even with a method capable of generate automaticly every test case of an application, the execution of all of them would be almost impossible in terms of time, and cost, therefore software testing technics can't ensure the absence of defects. Due to the high cost of the development and execution of software testing a way of improve the efficiency of this task is needed. An algorithm for generating test cases in an authomatic manner from UML specification and a set of practices oriented to rank these test cases in terms of the associated risk are proposed in this paper.

Resumen: La generación sencilla de los casos de pruebas de una aplicación ha sido a lo largo de la historia una de las principales aspiraciones de los desarrolladores de software. Sin embargo, en el mejor de los casos, aquel en el que dispusieramos de un método para la obtención automática de todos los casos de prueba, la ejecución real de dichos casos sería poco menos que inviable en términos de tiempo y coste por tanto las pruebas del producto no permitirán nunca asegurar la ausencia de defectos en el mismo ya que la complejidad de software hace inviable la realización de un control definitivo, exhaustivo y completo. Dado que los costes de aplicación de pruebas pueden ser muy elevados es necesario incrementar la eficiencia de dichas tarea. En este artículo se propone un algoritmo para la generación automática de casos de prueba a partir de especificaciones UML y una serie de prácticas orientadas a la ordenación de dichos casos de prueba en función del riesgo asociado en cada uno de ellos.

Palabras Clave: Calidad, Precedencia de Casos de Prueba, Especificaciones UML.

1. INTRODUCCIÓN

Las técnicas y procesos para el aseguramiento de la calidad son bien conocidos y, además, son parte de las buenas prácticas recomendadas por los modelos de evaluación y mejora de procesos en sus primeros niveles: [CMM Product Team, 2002a] [CMM Product Team, 2002b], proceso básico en ISO 15504 SPICE [ISO, 1999]. En general, las técnicas que más frecuentemente, con buenos resultados, se utilizan en el aseguramiento de calidad de software se corresponden con la medición de software, los procesos de revisión y auditoría y las pruebas de software (estas dos últimas encuadradas como principales técnicas en lo que se suele conocer como verificación y validación) Como indica el estándar IEEE 730 de aseguramiento de calidad [IEEE, 1998], estas técnicas componen el núcleo de actuación en el proyecto dentro de un buen entorno con adecuada gestión de configuración.

Dado que los costes de aplicación de revisiones, inspecciones y pruebas pueden ser muy elevados es necesario incrementar la eficiencia de dichas tareas, más aún cuando, a lo largo del tiempo, se ha evidenciado que las pruebas del producto no permitirán nunca asegurar la ausencia de defectos en él, ya que la complejidad del software hace inviable la realización de un control definitivo, exhaustivo y completo.

Centrando nuestro estudio en la realización de pruebas, en este artículo proponemos, no solo un algoritmo para la generación automática de casos de

prueba a partir de especificaciones UML, que ya de por sí hace el proceso más eficiente, sino que además se explica como utilizar técnicas de análisis de riesgos que permiten la ordenación o el establecimiento de una precedencia de los casos de prueba a partir de características propias de los requisitos del sistema que puede ser utilizada para decidir hasta donde debe llegar el equipo de pruebas para probar aquello considerado de alguna manera más prioritario y conseguir así cierto nivel de confiabilidad.

De acuerdo con el estándar NASA-STD-8719.13-A [NASA, 1997] el riesgo es una función de la frecuencia de ocurrencia de un evento no deseado y la seriedad potencial de las consecuencias derivadas del mismo, por lo tanto el riesgo asociado con un determinado requisito debe medirse en términos de la probabilidad de que ocurra un error relacionado con dicho requisito y el coste asociado a dicho error. De la misma manera y en relación con el desarrollo de software, en [Hutcheson, 2003] se habla de la necesidad de determinar con antelación la probabilidad de que ocurra un evento que cause algún tipo de daño y la severidad de dicho daño en términos de coste económico o humano (en el caso de que exista la posibilidad de perder vidas humanas). Sin embargo a menudo es imposible determinar con exactitud ambos valores y es necesario realizar una estimación a partir de un análisis cualitativo con el objetivo de descubrir qué afecta al riesgo y de un análisis cuantitativo que establezca en qué medida afecta cada uno de los puntos determinados previamente.

Hasta hace unos años, el análisis de riesgos estaba centrado en técnicas heurísticas de análisis como las utilizadas en la antigüedad por los filosofos griegos o las desarrolladas ya en el siglo 20 en [Polya, 1982]. En [Bach, 1999] se habla del método heurístico como una manera útil de encontrar la solución a un problema que no siempre funciona. En concreto y a la hora de probar un software se menciona la necesidad de preguntarse, para cada parte del producto, acerca de sus:

- Vulnerabilidades, en el sentido de debilidades del producto.
- Amenazas, como entradas o situaciones que pudieran provocar un fallo en el sistema.
- Victimas, es decir, quién o qué puede verse afectado por un fallo y como de malas serían sus consecuencias

En definitiva, el modelo heurístico proponía la elaboración de unas listas de riesgos que en su versión más moderna consistía en rellenar una tabla con tres columnas especificando componente, riesgo asociado basado en tres valores (Normal, Alto y Bajo) y heurística del riesgo o motivación de dicha catalogación (tabla 1)

Component	**Risk**	**Risk Heuristics**
Printing	Normal	distributed, popular
Report Generation	Higher	new, strategic, third-party, complex, critical
Installation	Lower	popular, usability, changed
Clipart Library	Lower	complex

Tabla 1. Matriz de riesgo de componentes

No es una mala técnica y es indudable que produce buenos resultados, pero como se dice en [Bach, 1999], debemos tener en cuenta que cualquier análisis de riesgo va a ser, de alguna manera, incompleto o impreciso por lo que queda patente la necesidad de utilizar, en combinación, otros métodos que mejoren nuestro nivel de acierto.

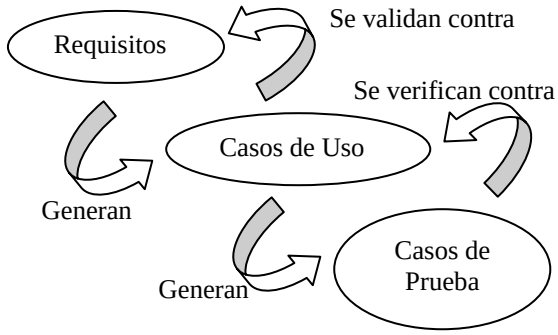


Figura 1. Relación entre Requisitos, Casos de Uso y Casos de Prueba.

Si se intenta aplicar el análisis de riesgos al desarrollo de aplicaciones orientadas a objetos y en concreto a la validación y verificación de dichas aplicaciones se debe hablar necesariamente de especificación de requisitos, casos de uso y casos de prueba. Según [Evans, 2002] los casos de uso deben probarse contra los requisitos, constituyendo este proceso la validación del sistema, mientras que son los propios casos de uso los que deben utilizarse como autoridad de prueba para probar el código, es decir para realizar la verificación del sistema (figura 1)

En este artículo se propone un proceso de generación y ordenación en función del riesgo de

casos de prueba orientados a cubrir las pruebas de caja negra asociadas a cada caso de uso. Esta investigación se enmarca dentro de otra de más alto nivel que pretende mejorar la calidad de los desarrollos web a partir de especificaciones UML, atacando el problema desde tres puntos de vista:

- Mejora de la calidad a través del estudio de la usabilidad del interfaz gráfico de usuario [Escribano, 2004]
- Mejora de la calidad a través del estudio del rendimiento del Servidor Web en ejecución [Villalba, 2004]
- Mejora de la calidad a través del desarrollo de un algoritmo para la obtención y precedencia de casos de prueba.

Este último punto es el que se tratará en este artículo.

2. PROCESO DE DESARROLLO PROPUESTO PARA LA GENERACION DE CASOS DE PRUEBA

Tanto el algoritmo de generación de casos de prueba, como las prácticas orientadas a la ordenación en función del riesgo y la propensión a fallos que se presentan en este artículo forman parte de todo un proceso, basado en el comentado en [Fernández et al., 2004] que desde las primeras etapas del diseño del software permite incorporar la filosofía de aseguramiento y medición de calidad productivos en desarrollos en Java.

La implementación del proceso global, se corresponde con parte de la figura 2 y consiste en la inclusión en las fases de diseño e implementación de una serie de controles o actividades de protección encargadas de asegurar en cierta medida la calidad de los resultados obtenidos en la fase de pruebas.

De esta manera, en resumen, el proceso se compondría de:

- Una fase de análisis que entre otras cosas obtiene los diferentes casos de uso analizando la perspectiva que el usuario tiene de la aplicación siguiendo la propuesta de [Fernández et al, 2003].
- Una fase de diseño, en la que además de producir resultados clásicos en la orientación a objetos como son los diagramas de clases, de colaboración y el diseño de los casos de prueba, se produce también una matriz o cubo tridimensional de trazabilidad entre Casos de Uso, Clases y Casos de Prueba a partir de la cual:

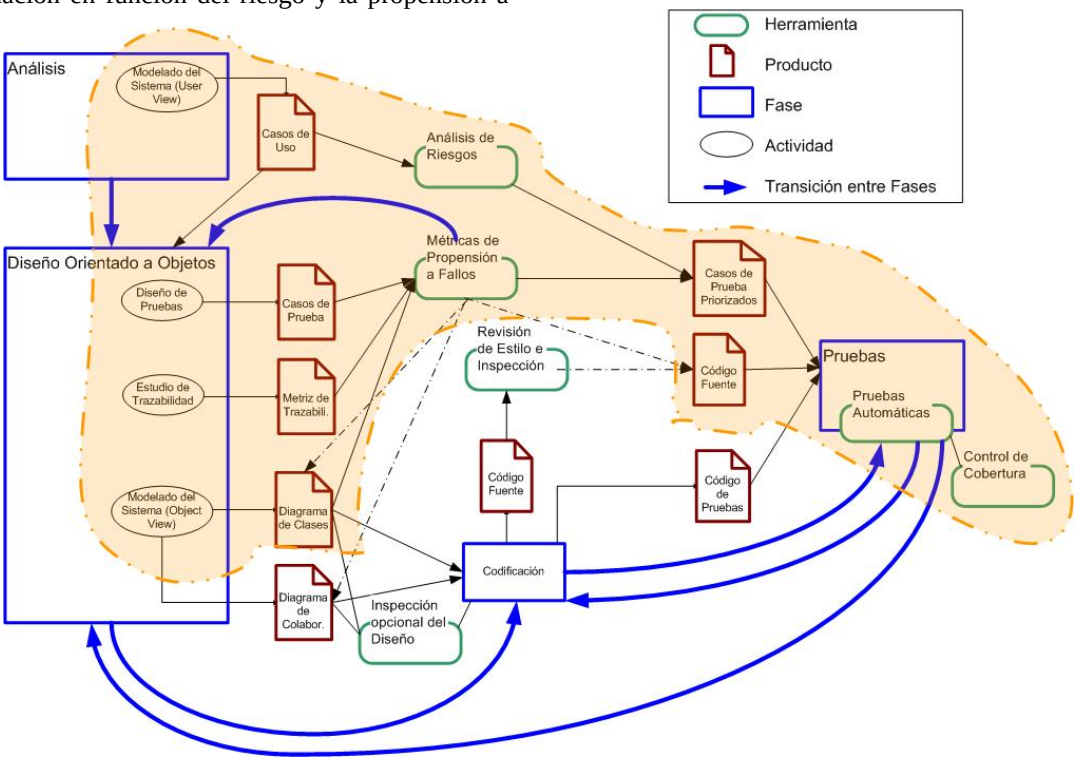


Figura 2. Proceso propuesto

- Utilizando herramientas de asignación de riesgos, como las presentadas en [Wang et Al., 2003] y [Hutcheson, 2003], basadas en la metodología de asignación de riesgos en función de diferentes artefactos UML descrita en [Goseva-Popstojanova et al, 2003], establecer un ranking de Casos de uso en función del riesgo.
- Utilizando como entradas algunos de estos resultados, aplicaríamos herramientas de medida que evaluasen, a través de ciertas métricas la propensión a tener defectos de cada clase [Chidamber y Kemerer, 1994] [Abreu y Melo, 1996] o [Briand et Al., 1997].

- Establecer un ranking de casos de prueba ordenándolos en función del nivel de riesgo de los casos de uso asociados a ellos (lo cual implica así mismo el estudio y análisis de cada requisito ínvolucrado en el origen del caso) y de la propensión al defecto de las clases relacionadas con ellos en nuestro cubo de trazabilidad.

- Ciclar en las fases de Ejecución de Pruebas, Rediseño y Remodificación tantas veces como sea necesario para conseguir el nivel deseado de defectos detectados. Se combinaría esta evaluación de pruebas con la aplicación de las más tradicionales medidas de cobertura.

A continuación detallaremos las técnicas que utilizaremos para la signación de riesgos así como las métricas utilizadas para la caracterización de aquellas clases que serán más propensas a fallar. Finalizaremos el artículo introduciendo brevemente el algoritmo de generación automática de casos de prueba.

2.1 Asignación de Riesgo

Una de las técnicas más novedosas a la hora de medir y asignar un valor de riesgo a una pieza de software, la cual será utilizada en nuestros futuros experimentos, es la conocida como MIT Risk Analysis o Análisis de Riesgo basado en las “Most Important Things” definido en [Hutcheson, 2003]. Según esta técnica deben utilizarse los siguientes criterios para establecer el riesgo de un determinado problema:

- Si afecta o no a la consecución de un **requisito**
- **Severidad** del problema.
- **Probabilidad** de que ocurra.
- **Coste** asociado al fallo
- **Visibilidad** del mismo
- **Tolerabilidad** del fallo por parte de los usuarios (humanos u otros sistemas)
- **Factor Humano**: Si supone o no que el usuario (humano) no consiga hacer lo que pretendía.

Cada uno de estos criterios sera aplicable o no a determinados problemas y en ese caso, y sólo en ese caso, recibirá una puntuación entre 1 y 5. Para lo cual el experto o grupo de expertos deberá contestar una serie de preguntas asociadas con cada criterio que permitirá posteriormente al análista decidir como valorarlo. El riesgo asignado a dicho problema vendrá constituido por la media de todos ellos.

Dado que nuestro objetivo final es ser capaz de decidir qué pruebas realizar para obtener el máximo rendimiento en nuestro esfuerzo de pruebas, se proponen además las siguientes categorías de problemas dentro del conjunto total de las MITs.

- Problemas no analíticos (MINs)
- Problemas relacionados con caminos más importantes (MIPs)
- Problemas relacionados con datos más importantes (MIDs)
- Problemas relacionados con diferentes entornos de ejecución (MIEs)

Una vez clasificados todos los problemas potenciales, se propone especificar para cada uno de ellos cuál es la cantidad necesaria de tests a realizar para considerar el problema suficientemente probado. A partir de ellas el número total de test para el total de los potenciales problemas (MITs) se corresponderá con la siguiente ecuación:

$$MITs = (MINs+MIPs+MIDs) \times MIEs$$

En nuestro caso y dado que nuestro algoritmo de generación únicamente explora caminos y datos y que hemos reducido el entorno de prueba a uno muy determinado consistente en pequeñas aplicaciones Java independientes, nos permitiremos simplificar la ecuación de la siguiente manera:

$$MITs = MIPs+MIDs$$

Una vez que tenemos todos estos números se puede calcular el número de ellos que debemos probar para reducir el riesgo por debajo de un determinado nivel con los siguientes dos métodos.

El primero calcula el número mínimo de test a realizar y simplemente debe dar una idea de hasta donde negociar a la hora de reducir los costes en pruebas. Siendo RM el riesgo medio de todos los MITs. Dicho valor se calcula con la siguiente fórmula:

$$\text{Minimum MITs} = \text{MITs} / \text{RM}$$

El segundo afina más el resultado, ya que tiene en cuenta además el número de test necesarios para descartar cada uno de los problemas y se calcula con la formula

$$\text{MITs} = \sum \text{minMITS(Ti))}$$

Siendo minMITS(Ti) el total de tests de cada problema potencial divido entre el riesgo asociado a dicho problema.

2.2 Determinación de la propensión al fallo de las clases

Hace ya algunos años que [Briand et al., 1998] sentaron las bases teóricas para el desarrollo de modelos que relacionaban ciertas métricas del software con medidas de la calidad del mismo que más tarde [Benlarbi, 2000] resumían en la figura 3. En ella se entiende por complejidad cognitiva la habilidad de comprensión de ciertas piezas de software por parte de los individuos que deben tratar con ellas, como desarrolladores, ingenieros de pruebas, inspectores de código o encargados del mantenimiento posterior. Un exceso de complejidad deriva, según dichas investigaciones en la aparición de efectos no deseados en características externas del producto como propensión a fallos del mismo o reducción de su mantenibilidad.

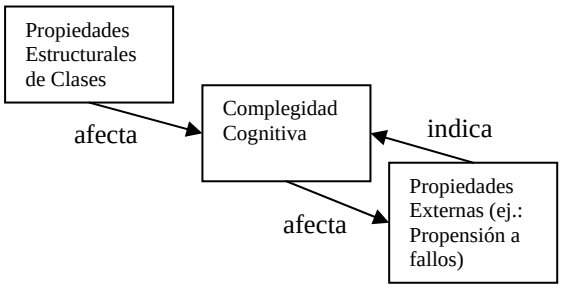


Figura 3. Bases para el desarrollo de metrcas de SW Orientado a objetos.

Estudios posteriores se centraron en adaptar, por un lado, alguna medida clásica como el tamaño del código fuente [El Emam et al., 2002] o el acoplamiento y la cohesión [Chidamber y Kemerer, 1994] [Henderson-Sellers, 1996] a la programación orientada a objetos y por otro en aislar y medir aquellas características especiales de la orientación a objetos como la herencia [Chidamber y Kemerer, 1994][Lorenz y Kidd, 1994] y el polimorfismo [Benlarbi, POLI].

A pesar de que [Benlarbi, 2000] no llegaron a encontrar en su investigación unos valores umbral para alguna de las áreas anteriormente descritas, a saber, tamaño, acoplamiento y herencia, diversos estudios han probado que dicha relación existe, aunque, tal y como el mismo Benlarbi asegura de una manera continua y no marcdamente diferente a partir de un determinado umbral.

Basándose en lo definido por [Archer y Stinson, 95], [García y Harrison, 2000] describen un conjunto representativo de métricas orientadas a objetos, cuyo correlación con número de defectos o esfuerzos de mantenimiento ya ha sido validada [Abreu y Melo, 1996] y [Basili et Al., 1995]. De hecho se pueden establecer determinados rangos de valores fuera de los cuales una clase o pieza del código es más propensa a errores. Podemos contar así con:

- Métricas a Nivel de Sistema: definidas por [Abreu y Melo, 1996] y conocidas como MOOD incluyen las conocidas medidas de proporción de métodos ocultos (MHF) y de métodos heredados (MIF) (sobre la base del total de métodos del sistema), proporción de atributos ocultos (AHF) y de atributos heredados (AIF) (sobre la base del total de atributos del sistema), proporción de polimorfismo

(PF) (relación del numero de métodos de una clase redefinidos en sus clases derivadas y el número total de métodos de las clase padre por el número de descendientes) y Proporción de Acoplamiento (CF) (relación entre número de relaciones entre clases excepto las de herencia y el numero total de pares de clases).

- Métricas de Acoplamiento: con la métrica de Acoplamientro entre clases de objetos (CBO) definida por [Chidamber y Kemerer, 1994] que mide el nivel de dependencia de unas clases con otras.
- Métricas de Herencia: un uso excesivo de la herencia puede ser peligroso para la calidad [Cartwright y Shepperd, 1996]. Algunas de las métricas mas representativas son la de Profundidad del Árbol de Herencia (DIT) y el número de hijos (NOC) definidas por [Chidamber y Kemerer, 1994] y orientadas a la cuantificación del árbol de herencia. También interesa el índice de Especialización por clase (SIX), que muestra en qué medida las subclases redefinen el comportamiento de sus superclases [Lorenz y Kidd, 1994].
- Métricas de Clases: estas métricas permiten detectar clases que necesiten ser redefinidas destacando ciertos aspectos de su nivel de abstracción. Por ejemplo, se pueden incluir la Respuesta de una Clase (RFC), definida por [Chidamber y Kemerer, 1994] (cuenta las ocurrencias de llamadas a otras clases desde una clase particular), los Métodos Ponderados por Clase (WMC) (complejidad de clase en función de la complejidad de sus métodos [Chidamber y Kemerer, 1994]) y la Falta de Cohesión en los Métodos (LCOM) (número de atributos comunes usados por diferentes métodos) de la que existen dos medidas propuestas [Chidamber y Kemerer, 1994] [Henderson-Sellers, 1996].

A estas podemos añadir las definidas por [Benlarbi, POLI] en relación a la medida del polimorfismo, que tal y como se concluye en su estudio afecta también a la propensión a fallo de una clase. Dichas métricas serían las siguientes: Sobrecarga en Clases Independientes (OVO), Polimorfismo estático en Predecesores (SPA), Polimorfismo estático en Descendientes (SPD), Polimorfismo dinámico en Predecesores (DPA) y Polimorfismo dinámico en Descendientes (DPD)

2.3 Algoritmo de generación de casos de prueba a partir de especificaciones UML

Toda la capacidad de detección de defectos de las pruebas se basa en la consideración de cualquier tipo de discrepancia entre la salida obtenida y la salida esperada (según lo indicado en las especificaciones) como síntoma de un problema en el software. En consecuencia, todas las técnicas de diseño de casos de prueba existentes se basan en tomar las especificaciones como referencia de comportamiento de la aplicación.

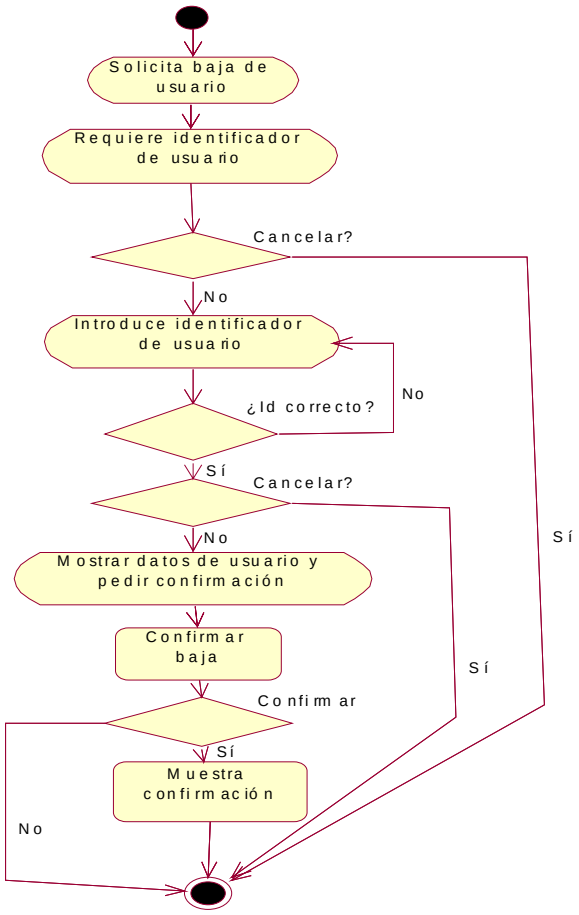
Por supuesto, el aumento en la complejidad y sofisticación tecnológica del software ha supuesto la evolución de estas reglas básicas de diseño y, sobre todo, de las técnicas de especificación y de diseño de software. Así, la llegada de las interfaces de usuario gráficas y las aplicaciones cuyo funcionamiento está basado en eventos complicó el manejo en las pruebas de las combinaciones de entradas y de las opciones de funcionamiento.

Sin embargo, las pruebas deben seguir basándose en las especificaciones que sirven de referencia de las necesidades del cliente. Desde hace mucho tiempo (por ejemplo, [Varios, 1983]), se ha optado por especificaciones de las interfaces gráfica de usuario (GUI) que describen su comportamiento a través de diversas variantes de diagramas de estados que reflejan la evolución dinámica de su funcionamiento y del flujo de eventos implicado. Así ocurre, por ejemplo, en [Memom, 2001], mientras que otras propuestas se centran en los diagramas de estados que describen el comportamiento de cada clase [Vieira, 2000]. De hecho, cualquiera de estas representaciones permite describir criterios de cobertura para la GUI o la estructura de la aplicación así como métodos de diseño sistemático de las pruebas o de verificación a partir de dichos modelos de especificación.

En nuestra propuesta deseamos aproximarnos a enfoques metodológicos con un mayor grado de implantación en los proyectos de desarrollo profesional y comercial. Así, la implantación de UML [Booch, 1999] como notación de desarrollo

orientado a objetos ha permitido la difusión e implantación real de técnicas de especificación de software como los casos de uso.

Como especificación los casos de uso resultan especialmente aclaratorios cuando se complementan con diagramas de actividad (figura 4). Un diagrama de actividades es un caso especial de un diagrama de estados. Éste último puede definirse como un modelo que muestra el conjunto de estados por los cuales pasa un objeto durante su vida en una aplicación, junto con los cambios que permiten pasar de un estado a otro. En los diagramas de actividad, casi todos los estados son estados de acción (identifican qué acción se ejecuta al estar en él) y casi todas las transiciones son activadas al terminar la acción ejecutada en el estado anterior.



Ya existen desarrolladas técicas que utilizan los diagramas de estado para validar un objeto mediante mecanismos de cobertura consistenetes en ejercitar todas las transiciones del diagrama de estados de un objeto con el fín de probar la implementación de este [Offut y Abdurazik, 1999]. Pero lo verdaderamente interesante en nuestro caso es que un diagrama de estados, o uno de actividad en nuestro caso, permiten detallar el comportamiento, no ya de un objeto, sino de un caso de uso completo y por lo tanto del reflejo de una o varias de nuestras especificaciones de partida.

Nuestra propuesta es adoptar un criterio de selección de casos de prueba basado en la exploración de los distintos escenarios de ejecución de caso de uso. En un diagrama de actividad, podemos asumir que existen diversos tipos de escenarios marcados por cada camino que se puede trazar en el mismo partiendo de su estado inicial. El número de caminos distintos en un diagrama de actividad suele ser muy elevado incluso en los casos más sencillos. Debemos tener en cuenta que un camino puede diferenciarse de otro simplemente en el hecho de que pase por un ciclo del diagrama de actividad un número distinto de veces. Pero no sólo el número de caminos diferentes de un diagrama de actividad puede ser muy elevado sino que dentro de cada camino (tipo de escenario) podemos distinguir múltiples escenarios de pruebas posibles, simplemente considerando todas las posibilidades de valores de datos de entrada posibles.

En nuestro caso, por tanto, el **primer criterio** de elección de casos se basará en la traza de caminos a través del diagrama de actividad. Será un criterio de cobertura similar al que se plantea con otros tipos de diagramas en [Memom, 2001]. Para ello nos inspiraremos en la teoría de grafos y utilizaremos un criterio análogo al que [McCabe, 1976] utilizaba con los grafos de flujo de control del código procedimental y podríamos expresarlo de la siguiente manera:

“Un caso de uso, a través de su diagrama de actividad, estará suficientemente probado si los caminos utilizados (tipos de escenarios) permiten pasar por todas las flechas o transiciones del diagrama de actividad al menos una vez.”

Sin embargo esto no es suficiente ya que en cada camino de un diagrama de actividad (tipo de escenario) pueden existir demasiados escenarios

distintos como para asegurar la representatividad de todos ellos en un solo caso de prueba. Siguiendo las recomendaciones tradicionales de diseño de pruebas de caja negra [Myers, 1979], deberíamos realizar un análisis del dominio de valores de entrada para detectar grupos/clases de datos que cuenten con un comportamiento homogéneo en el sistema; es decir, debemos utilizar la técnica conocida como de clases de equivalencia.

Por lo tanto, nuestro segundo criterio de cobertura de pruebas basadas en casos de uso se expresará como sigue [Myers, 1979]:

"Para las entradas de datos en un caso de uso, debemos lograr que todas las clases de equivalencia válidas sean ejercitadas por un caso de pruebas y que cada clase no válida sea comprobada en solitario en un caso específico, logrando que se ejerciten todos los valores límite".

La combinación de ambos criterios en un diagrama de actividad resulta fácil utilizando los elementos de representación contemplados en la notación UML. Así, en los pasos de un caso de uso que supongan entrada de datos, no utilizaremos el concepto de acción sino el de actividad. Mientras que una acción debe ser elemental y no puede ser interrumpida cuando se inicia [Jacobson, 1992], una actividad "invoca un subdiagrama de actividad. Cuando se entra en un estado de actividad, el diagrama anidado se ejecuta [...] No se sale del subdiagrama hasta que se alcanza su estado final" [OMG, 2001].

En nuestro caso, cada paso de introducción de datos se convertirá en un estado de actividad cuyo subdiagrama deberá expresar las distintas opciones de prueba basadas en clases de equivalencia y valores límite (figura 5).

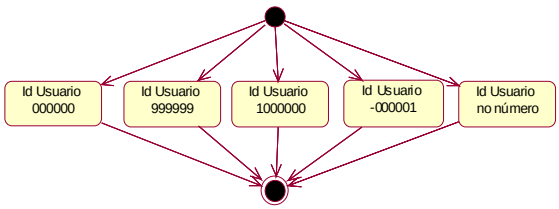


Figura 5. Subdiagrama de actividad para Introducir ID de la Figura 5.

Contemplando los dos criterios de cobertura expresados en nuestra propuesta, el método de generación de casos de pruebas se realizaría en la práctica de la siguiente manera:

1. Generar para cada caso de uso, a partir de la descripción de su interacción (ver apartado 2), un diagrama de actividad primario donde se marcan los caminos de flujos de eventos (figura 2).
2. Identificar en cada flujo la introducción de datos y analizar para cada caso las clases de equivalencia, valores límites e, incluso, tratamiento de combinaciones (como el previsto en [Fernández, 2000] tomado de [Myers, 1979]).
3. Transformar cada acción de entrada de datos en un estado de actividad que contenga como subdiagrama las opciones de entradas y de tratamiento de las mismas (figura 5), generando el diagrama definitivo (figura 4).
4. Contemplando el diagrama definitivo, cumplir con los dos criterios de cobertura de manera que todas las flechas o aristas del mismo sean ejercitadas, al menos, una vez.

3. CONCLUSIONES

Hemos presentado un proceso que combina la eficacia en la detección de defectos y el aseguramiento de calidad en el desarrollo de software con la eficiencia y la productividad en la aplicación del mismo. Este equilibrio permite optimizar los beneficios de la mejora de calidad conseguida a medio plazo con la minimización de costes necesarios para ello. Esta combinación permite reducir la resistencia del *management* en la promoción de esta filosofía a la vez que facilita que los desarrolladores se integren con facilidad en un proceso que evalúa y controla la calidad. En todo el proceso, la automatización y la adaptación a las circunstancias del proyecto y de la organización.

Además nuestra propuesta está inspirada también en la mejora de la eficiencia simultánea a la priorización y selección de intensidad de control. A modo de resumen terminaremos recordando que incluye los siguientes puntos:

- Creación simultánea de casos de prueba de aceptación a partir de la definición de requisitos e interacción funcional descrita en

UML como casos de uso complementados con diagramas de actividad [Fernández et al, 2003]. Esta posibilidad incentiva el cuidado de la especificación UML ya que, por el procedimiento descrito, se genera automáticamente un diseño bastante detallado de las pruebas de aceptación necesarias para cubrir la funcionalidad y la interacción solicitada.

- Diseño detallado de pruebas a partir de la información de diseño (como ya se ha descrito en el apartado 2.3 de este artículo) donde el análisis de Pareto del diseño colabora en el establecimiento de un clasificación de casos en función del riesgo para ayudar a decidir la posible intensificación selectiva de controles incluida en nuestra filosofía de proceso.
- Ejecución de pruebas priorizada en función del riesgo final aceptable para el proyecto y el tiempo y los recursos disponibles. Aparte de la información de priorización obtenida de las anteriores etapas las buenas prácticas de pruebas recomiendan el control de la eficacia de las mismas. La manera más habitual y sencilla de lograrlo es el control de cobertura de la ejecución de pruebas. En nuestro caso, hemos contado con la ayuda del módulo TestChecker de Logiscope que permite controlar si las pruebas han pasado por todas las sentencias, todas las opciones de decisiones (caminos de decisión a decisión: DDP), etc. Esta información se puede obtener tanto de forma gráfica como valores estadísticos. Así, la gestión del proyecto puede decidir si completar el 100% de la cobertura de sentencias (como es habitual en proyectos para la Agencia Espacial Europea por normativa) o, dadas las circunstancias, llegar al 80% (asumiendo ciertos riesgos) o completar el 100% en componentes propensos a defectos y reducir el porcentaje al 70%, por ejemplo, en el resto. El proceso se resume en un ciclo que consiste en ejecutar casos seleccionados (por ejemplo, los generados desde el análisis y el diseño), comprobar la cobertura y añadir nuevos casos según se requiera para completar el criterio de pruebas.

4. REFERENCIAS

Archer C. y Stinson, 1995, M. Object Oriented software measures, Technical Report CMU/SEI-95-TR-002, abril 1995.

Basili V.R., Briy L. y Melo W., 1995 A Validation of OO design metrics as quality indicators, Technical report CS-TR-3443, 1995.

Benlarbi S., Melo W. L., 1999, International Conference on Software Engineering. Proceedings of the 21st international conference on Software engineering.

Benlarbi S, El Emam K, Goel N, Rai S, 2000 Institute of Information Technoloy. National Research Council, Canada.

Bolour A, 2003, Notes on the Eclipse Plug-in Architecture, Bolour Computing. (http://www.eclipse.org/articles/index.html)

Booch, G., Rumbaugh J. y Jacobson, I., El lenguaje unificado de modelado, Addison-Wesley, 1999.

Briand L., Devanbu, P. Melo, W, 1994, An investigation into coupling measures for C++, In Proceedings of the 19th International Conference on Software Engineering.

Briand L., Wuest, J. Iconomovski, S, 1998, A comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study, International Software Engineering Research Network, Technical Report ISERN-98-29.

Brito e Abreu F. y Melo W., 1996, Evaluating the impact of object oriented design on software quality, Proceedings of 3rd international software metrics symp.

Cartwright M. y Shepperd M, 1996, An empirical investigation of object oriented software in industry, Department of Computing, Talbot Campus, Bournemouth Uiversity, Technical Report TR 96/01.

Chidamber S.R. y Kemerer C.F., 1994, A metricsuite for object oriented design, IEEE transactions on Software Engineering, vol. 20, nº 6, junio, pp 467-493.

CMMi Product Team, 2002, CMMISM for Software Engineering, Version 1.1, Continuous Representation (CMMI-SW, V1.1, Continuous) CMU/SEI-2002-TR-028, Software Engineering Institute.

CMMi Product Team, 2002, CMMISM for Software Engineering, Version 1.1, Staged Representation (CMMI-SW, V1.1, Staged) CMU/SEI-2002-TR-029, Software Engineering Institute.

Escribano J.J., Lara P.J., Villalba M.T., Fernández L., 2004. Use Case for enhancing IS

requirements management. IADIS International Conference e-Society 2004.
Fernández, L., 2000 “Utilización de casos de uso en las pruebas de aceptación”, V Jornadas sobre Calidad del Software, 2000, pp. 65-76.
Fernández, L., Lara, P.J., Gutiérrez, C., 2003, Actas de las VIII Jornadas de Innovación y Calidad del Software, ATI, pp. 26-32.
Fernández, L., Lara, P.J., 2004, Proceso Y Herramientas Para La Productividad En El Aseguramiento Y Medición De Calidad En Desarrollos Java, Revista de Procesos y Métricas, AEMES, Nº2, Agosto 2004
Garcia D, Harrison R., 2000, “Medicion en la Orientación a Objetos”en L. Fernadez y J. Dolado, Medición para la gestión en la Ingeniería del Software, Ra-Ma, pp. 75-92.
Goseva-Popstojanova K., 2003, Architectural-Level Risk Analysis Using UML, IEEE transactions on software engineering, vol. 29, nº 10, octubre 2003, pp. 946-960.
Henderson-Sellers B., 1996, Object Oriented metrics measurements of complexity, Prentice-Hall.
Hutcheson M.L., 2003, Software Testing Fundamentals, Wiley Publishing Inc. 2003.
IEEE, 1998, IEEE Std. 730-1998. Standard for Software Quality Assurance. IEEE Computer Society.
ISO, 1999, Information Technology. Software process assessment. Part 2. A reference model for processes and process capability, ISOIEC Tr 15504-2, ISO.
Jacobson, I., Object-oriented software engineering, Addison-Wesley, 1992.
Lara P.J., Fernández L., 2004, ¿Cómo mejorar el estilo en desarrollos Java?, Dr Dobbs Journal España, abril 2004, pp 54-59
Lorenz M., Kidd J., 1994, Object Oriented Metrics, Prentice-Hall.
Myers, G.J., 1979, The art of software testing, John Wiley & sons.
Memon, A.T., Soffa, M.L. Pollack, M.E. 2001, “Coverage criteria for GUI testing”, 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (FSE-9).
McCabe T., 1976, A complexity Metrics, IEEE transactions on software engineering, vol. 2, nº 4, diciembre, pp. 308-320
NASA, 1997, NASA Technical Standard 8719.13A, Software Safety.
Offutt J., Abdurazik A., 1999. Generating Tests from UML Specifications. Second International Conference on the Unified Modeling Language (UML99), pages 416-429, Fort Collins, CO, October 1999.
OMG, 2001, Unified Modelling Language Specification, version 1.4.
Polya G, How to Solve It: A New Aspect of Mathematical Method Princeton Univ Pr; Reissue edition, Enero 1982
Varios, 1983, JSP [and] JSD : the Jackson approach to software development, editado por J.R. Cameron, IEEE Computer Society Press, 1983
Villalba M.T., Fernández L., Escribano J.J, Lara P.J., , 2004. Un Estudio sobre rendimiento Web. Conferencia IADIS Iberoamericana WWW/Internet 2004.
Vieira, M.E., Dias M.S. y Richardson, D.J., 2000, “Object-oriented specification-based testing using UML statecharts diagrams”, Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000), junio, 2000.
Wang T., Hassan A., Guedem A. Abdelmoez W., Goseva-Popstojanova K, Ammar, H., 2003, Architectural level risk assessment tool based on UML Specification, Proceedings of the 25th International Conference on Software Engineering (ICSE’03)