Procedimiento para la realización de pruebas de unidad de software orientado por objetos a nivel de clases

Procedure for unit (class) testing of object oriented software

Mauricio Alba Castro. M.Sc.
Departamento de Ciencias Computacionales, Universidad Autónoma de Manizales, Colombia
malba@autonoma.edu.co

Recibido para revisión 02 de diciembre de 2010, aceptado 28 de junio de 2011, versión final 30 de junio de 2011

***Resumen*— El artículo describe un proceso de pruebas de unidad, a nivel de clases, de software orientado por objetos. El objetivo del proceso de pruebas es medir la probabilidad de fallo de cada clase. La probabilidad de fallo de una clase puede predecirse en función de algunas métricas de tamaño y complejidad utilizando un modelo logístico. La probabilidad de fallo de las clases permite planificar el esfuerzo de pruebas de manera que dedique más tiempo de pruebas a las clases cuya probabilidad sea mayor. El diseño de los casos de prueba combina tres técnicas de caja negra: la técnica de la partición equivalente, la técnica de los valores límite y la técnica de la matriz ortogonal. El proceso ha sido validado probando algunos métodos de algunas clases. Se desarrolló un prototipo de software que asiste en el diseño de los casos de prueba.**

***Palabras clave*— Pruebas de unidad OO, Probabilidad de fallo, Métricas orientadas por objetos, Modelo logístico.**

***Abstract*— The paper describes a unit testing procedure for object oriented software at class level. The goal of the testing procedure is to measure each class fault-proneness. The class fault-proneness can be predicted as a function of some object-oriented software metrics, including class size and complexity, by using a logistic model. The class fault-proneness can be used during testing effort planning, in order to focus the testing effort in those classes whose fault-proneness estimation is higher. The testing cases design combines three black box techniques: the equivalent partition technique, the extreme values technique and the orthogonal array technique. The procedure has been validated by testing some methods of some classes. A software prototype was developed to assist in the design of the testing cases.**

***Key words*— OO unit testing, fault-proneness, Object-oriented metrics, Logistic model**

I. INTRODUCCIÓN

El proceso de pruebas de unidad del software, en el caso de la programación orientada a objetos, se realiza a nivel de clases para medir de forma consistente la probabilidad de fallo de cada una de las clases que están incluidas en el software [14], sin importar su tamaño u otras características, que también son medidas [12, 13]. En un proceso de desarrollo de software, los pronósticos acerca de la probabilidad de fallo de las clases que se programan, son utilizados en la gestión de las actividades de testing, para fijar prioridades y enfocar el trabajo de las pruebas en aquellas clases con mayor riesgo de fallo. Los pronósticos acerca de la probabilidad de fallo de las clases, también han sido utilizados para pronosticar las solicitudes de servicio post-entrega del software, y así el esfuerzo y costo de los servicios post-entrega del software. Esta probabilidad de fallo, medida como el porcentaje de las pruebas realizadas y no aprobadas [14], así como otras métricas orientadas por objetos, medidas en el código fuente, serán utilizadas en la validación del modelo logístico que predice ésta probabilidad en función de las otras medidas como el tamaño de la clase (número total de líneas de código fuente y número de métodos), la complejidad de la clase, entre otras [2, 12, 13]. El modelo logístico es utilizado en [10], [1] y [5].

No se incluyen pruebas de aceptación ni pruebas de sistema, pero en el caso de clases que usan otras clases que también deben probarse, se incluyen las pruebas de integración de la clase cliente con las clases servidoras. En otras palabras, se probarían inicialmente las clases localizadas en la capa inferior del software y luego se probarían las de la siguiente capa hacia arriba, en un proceso de abajo hacia arriba ("bottom-up"), similar a un proceso de pruebas de integración ascendente [6]. Con éste orden se espera facilitar la realización del proceso de pruebas empezando por las clases más simples (solo involucran objetos de una sola clase: ella misma, tienen pocos métodos,

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

o ninguno, con parámetros que sean objetos de alguna clase). Igualmente, permite ir conociendo poco a poco el software que se está probando, también empezando por las clases más simples. El proceso describe la forma de construir y especificar los casos de prueba de cada clase, empleando principalmente técnicas de caja negra (solo se conocen las entradas y salidas y sus especificaciones [8]), pero con el conocimiento de algunas características de la implementación. Se parte de la especificación de las clases y sus métodos públicos, pero también se usan los tipos de los atributos o campos de la clase y el encabezado, o prototipo, de la implementación de los métodos públicos (nombre y tipo del resultado, y los nombres y tipos de los resultados). Como no se requiere un proceso de desarrollo de software específico, las especificaciones de las clases y métodos pueden ser frases en español, diagramas de clases, diagramas de estado de las clases, así como las invariantes de las clases y las precondiciones y post-condiciones de los métodos públicos, descritos en español o en lenguaje OCL ("Object Constraint Language"), o en otros lenguajes formales de especificación propios de los lenguajes de programación (JML para Java).

El diseño de los casos de prueba de un método, combina en secuencia tres técnicas de caja negra. Inicialmente se aplican dos, la técnica de la partición equivalente y la técnica de los valores extremos, para seleccionar valores de prueba para cada variable, por separado. Posteriormente se aplica la técnica del arreglo o matriz ortogonal para seleccionar combinaciones de valores de las variables de la prueba del método. Esta técnica permite asegurar que se prueben todas las combinaciones con interacciones de dos de esas variables. La técnica de la matriz ortogonal se usa en el diseño de experimentos que aplica métodos de Taguchi. Esta técnica se viene aplicando con éxito al testing del software [7, 3, 4]. El proceso de pruebas de cada proyecto de desarrollo se lleva a cabo en cuatro fases, como lo ilustra la Figura 1, donde además se señalan los instrumentos que se usan en cada una de ellas [13]: i) La planificación general (Sección 1), ii) el diseño y especificación de los casos de prueba (Sección 2), iii) la planificación detallada de los casos de prueba (Sección 3) y finaliza con iv) la ejecución de los casos de prueba (Sección 4). En la Sección 5 se describe el trabajo experimental inicial realizado como prueba piloto del proceso de pruebas diseñado, y finalmente en la Sección 6 se concluye.

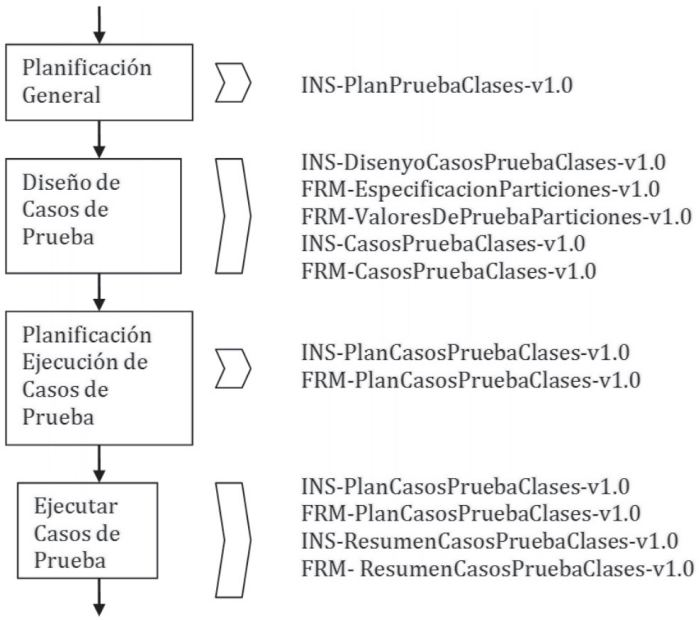

Figura 1. El proceso de pruebas

II. PLANIFICACIÓN GENERAL DE LAS PRUEBAS DE UN PROYECTO DE DESARROLLO DE SOFTWARE

En esta fase se especifican las clases que se van a probar, el orden en que se van a probar, las características que serán consideradas en las pruebas, también las que no serán consideradas, y por último las especificaciones de las clases que serán empleadas, entre otros: casos de uso en los que participan las clases, diagramas de clases, diagramas de estado de las clases (por ejemplo en notación UML), invariantes de las clases expresados usando texto narrativo o algún lenguaje formal (OCL, predicados matemáticos), contratos de las operaciones de las clases con las precondiciones y las post-condiciones (expresadas usando texto narrativo o algún lenguaje formal), el código fuente, comentarios en el código, aserciones y cláusulas propias del lenguaje de especificación disponible para el lenguaje de programación (por ejemplo JML para Java).

Idealmente se pretende probar en todas las clases desarrolladas, principalmente características correspondientes a los requerimientos funcionales, pero se podrían incluir características de requerimientos no funcionales como desempeño, usabilidad, etc. siempre y cuando no se requiera más de unos pocos días para establecer el escenario de las pruebas y según la disponibilidad de los equipos y personas de la empresa que puedan ser asignados al proceso de pruebas. El orden en que se van a probar las clases depende del diagrama de clases y de las colaboraciones existentes entre las clases. Inicialmente se pruebas las clases que no delegan responsabilidades en otras clases (no tienen colaboraciones, no invocan operaciones de otras clases, es decir que son clases servidoras netas). Luego se pasa a probar aquellas clases que tienen como colaboradoras a las anteriores, y así sucesivamente. En caso de recursión mutua entre dos clases (caso aún no presentado), hay que decidir cuál de las dos se implementa como una clase ficticia para probar primero la otra [6].

Una vez establecidas las características que serán consideradas en las pruebas, las especificaciones que serán utilizadas además del código fuente, clases a probar y el orden en que serán probadas se establecen de forma general los criterios de fallo [6] de cada una de las clases. Por ejemplo la no satisfacción por parte de los métodos que implementan las operaciones de las post-condiciones del método y los invariantes de la clase, o la llegada del objeto de la clase a un estado sin satisfacer el diagrama de estados (por hacer un cambio a un estado distinto o a uno no definido en el diagrama de estados). Finalmente se hace una planificación tentativa de las pruebas de todos los métodos de cada clase, planificación que será ajustada luego de haber especificado detalladamente los casos de prueba de cada método público de las clases que se van a probar, en la Sección 3.

III. DISEÑO DE CASOS DE PRUEBA

Este diseño comprende un proceso a nivel de cada clase, que se lleva a cabo para cada uno de los métodos públicos de la clase. El objetivo es obtener un conjunto representativo de casos de prueba para todos los métodos públicos, que sin embargo sea el más pequeño posible. Esto con el fin de que el tiempo requerido para realizar las pruebas sea el mínimo posible. La pruebas de un método deben considerar los datos de entrada (los parámetros del método), los datos de los campos del objeto (o clase) correspondiente, y un escenario con enlaces y objetos existentes, con datos en sus campos. El diseño de los casos de prueba determinará con qué datos se realizará cada caso de prueba.

Para iniciar el proceso con un método se deben identificar primero todas las variables de entrada de la prueba, que pueden ser parámetros del método, y campos de objetos y enlaces (variables de estado) relevantes. El diseño de los casos de prueba de un método, combina en secuencia tres técnicas de caja negra [8]. Inicialmente se aplican dos, la técnica de la partición equivalente y la técnica de los valores extremos, para seleccionar valores de prueba para cada variable por separado. Posteriormente se aplica la técnica del arreglo o matriz ortogonal para seleccionar combinaciones de valores de las variables de la prueba del método para asegurar que se prueben combinaciones con interacciones de al menos dos de esas variables. La técnica de la matriz ortogonal se usa en el diseño de experimentos que aplica métodos de Taguchi. Esta técnica se viene aplicando con éxito al testing del software [7, 3, 4].

3.1 La Técnica de la Partición Equivalente

La partición equivalente [6, 7] divide el conjunto de posibles valores tanto válidos como inválidos en sub conjuntos disyuntos entre sí (cuya unión colectiva forma el conjunto completo de posibles valores). Estos subconjuntos, particiones, son tales que, los diferentes valores de esos subconjuntos son equivalentes desde el punto de vista de la especificación del método. Se parte de la información del tipo de la variable (en el lenguaje de programación) y del conjunto de valores posibles del tipo. El conjunto de valores posibles se subdivide en subconjuntos disyuntos de valores válidos e inválidos (inicialmente uno de cada uno). Esos subconjuntos iniciales se subdividen según las especificaciones del método. El resultado para cada variable son una cantidad de particiones numeradas con sus conjuntos de valores.

Procedimiento

Para cada método público de cada clase:

1. Identificar los objetos participantes y los enlaces existentes entre ellos y su visibilidad desde el objeto activo (this o self).
2. Se identifican todas las variables que determinan el comportamiento del método, como son los parámetros de entrada del método, las variables de estado (los campos) y los enlaces entre los objetos participantes incluido el objeto activo. Deben incluirse las variables de clase, es decir las variables estáticas en lenguajes como Java.
3. Se particiona el dominio (conjunto de posibles valores según el tipo en el lenguaje de programación usado) de cada variable identificada en el paso anterior, en clases de valores equivalentes con base en la especificación y el comportamiento de la operación (método) y de la clase, así:

- Establecer el dominio de la variable (tipo y conjunto de valores).
- Particionar el dominio en dos conjuntos de 1) valores válidos y 2) valores inválidos.
- Particionar los conjuntos de valores válidos e inválidos de la variables con base en la especificación (Restricciones -invariantes, pre y post-condiciones, Diagramas de Estados de las clases, y el Contrato de la operación).
- Especificar las particiones usando el formato correspondiente.

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

A continuación se presenta la adaptación del ejemplo de prueba de un caso de uso, publicado como material de la asignatura Ingeniería de Software II del programa Ingeniería de Sistemas.

Ejemplo 1. Las Particiones Equivalentes de un método

Se tiene el siguiente contrato de la operación "Reservar Documento" de la clase Biblioteca:

Cuadro 1. Contrato de la operación "Reservar Documento" de la clase Biblioteca

Nombre	Reservar Documento
Clase	Biblioteca
Responsabilidades	Verificar que el código corresponda a un usuario registrado, no retirado ni sancionado (estado normal). Mostrar el Nombre completo del usuario. Verificar que el Número corresponda a un documento registrado que se pueda reservar. Mostrar el título completo del documento. Verificar que la fecha sea de un día hábil y que el usuario no tenga otra reserva para la misma fecha. Registrar la Reserva.
Parámetros	CódigoDeUsuario, NúmeroDeDocumento, FechaReserva
Salida	Objeto Reserva o valor NULL
Precondiciones	
Postcondiciones	Si las verificaciones son positivas: 1) se crea un objeto reserva con atributo fecha = FechaReserva, y atributo estado = activa. 2) se crean tres enlaces (asociaciones) del nuevo objeto con los objetos Biblioteca, Usuario y Documento correspondientes.
Excepciones	Si alguna de las verificaciones es negativa no se realiza la reserva, y retorna NULL. Debe mostrarse el mensaje correspondiente (Usuario no existe/Usuario Retirado/Usuario Sancionado, Documento no existe/ Documento no se puede reservar, Fecha No hábil/Tiene otra reserva misma Fecha).
Notas	El código del usuario es de 8 dígitos (el número de la cédula), y el número del documento es numérico de seis dígitos desde el 000001 al 010000. El formato para la fecha es: dd-mm-aaaa. Por ejemplo: 12-03-2008, para el 12 de marzo del 2008, y hay meses de 30 y 31 días, y Febrero puede tener 28,29 días.

Para ayudar a construir la tabla de partición de equivalencia, se elaborará una tabla (o un árbol) para cada una de las entradas, de izquierda a derecha, identificando grupos de valores equivalentes que se numeran y describen, diferenciando valores válidos e inválidos, según el formato de los datos de entrada, y de acuerdo con los estados de los objetos, las pre-condiciones y las post-condiciones y demás aspectos relevantes de las especificaciones, así:

Cuadro 2. Las Particiones de la variable CódigoDeUsuario

Variable Entrada:		CódigoDeUsuario	Tipo:	String
#	Válido/No Válido	Formato	Objeto/Variable Estado	
			Usuario	Estado
1	Válido	8 dígitos	Existe	Estado Normal
2	No válido	8 dígitos	Existe	Sancionado
				Retirado
3	"	8 dígitos	No Existe	
4	"	En blanco		
		No dígitos		
		No 8 dígitos		

Nótese que las dos columnas de la derecha no tienen que ver con los valores de la Variable Entrada. Esas columnas son relativas al objeto Usuario y su estado. Se debe establecer la correspondencia entre los estados y conjuntos de valores de algunas de las variables de instancia/clase de los objetos participantes en la colaboración correspondiente al contrato. En el ejemplo participan objetos Usuario, objetos Documento y objetos Reserva, visibles desde el objeto Biblioteca. Los objetos Reserva enlazan un objeto Usuario y un objeto Documento. El estado del objeto Usuario es el relevante para esta entrada CódigoDeUsuario. Las variables de instancia del objeto Usuario pertinentes son CódigoDeUsuario y Estado (Normal/Sancionado/Retirado)

Cuadro 3. Las Particiones de la variable NúmeroDeDocumento

Variable Entrada:		NúmeroDeDocumento		Tipo:	Integer
#	Válido/No Válido	Formato		Objeto/Clase	Variable Estado
		Tipo/Tamaño	Valores	Documento	Disponibilidad
1	Válido	Numérico de 6	En Rango	Existe	Disponible
2	"	"	En Rango	Existe	No disponible
3	No válido	"	En Rango	No Existe	
4	"	No numérico de 6			
5	"	Numérico de 6	Fuera de Rango		
		Numérico de otro tamaño o En blanco			

En el caso de la Variable de Entrada NúmeroDeDocumento, es relevante no solo el formato sino conjuntos específicos de valores definidos por los rangos de las notas del contrato. Por otra parte, el objeto relevante es el objeto Documento cuya variable NúmeroDeDocumento tiene el mismo valor que la entrada, y su Disponibilidad para ser reservado (Disponible/No Disponible), Los Disponibles pueden ser Reservados.

Cuadro 4. Las Particiones de la variable FechaReserva

Variable Entrada:		FechaReserva			Tipo:	DD/MM/AA
#	Válido/No Válido	Formato			Objeto/Clase	Variable Estado
		Tipo/ Tamaño	Valores		Reserva	FechaReserva
1	Válido	Formato correcto	En Rangos	Hábil	No Existe	Ese día para el mismo Usuario o Documento
2	"	"	En Rangos	Hábil	Existe	Ese día para el mismo Usuario o Documento
3	No válido	"	En Rangos	No Hábil		
4	"	"	Fuera de Rango			
5	"	Formato incorrecto				
6		En blanco				

En el caso de la Variable Entrada FechaReserva, las dos columnas de la derecha corresponden a las Reservas, específicamente a la (in)existencia de un objeto Reserva enlazado al mismo Usuario y con la misma fecha. En resumen, se tienen 3 variables de Entrada (CódigoDeUsuario, NúmeroDeDocumento, FechaReserva) y 3 variables de Estado de 3 objetos participantes (Usuario: Estado, Documento: Estado, Reserva: Fecha). Se tienen entonces, 4 particiones sobre dos variables, una variable de Entrada (CódigoDeUsuario) y una variable de Estado (CodigoDeUsuario:Usuario.Estado); 5 particiones sobre otras dos variables, una variable de Entrada (NúmeroDeDocumento) y una variable de Estado (NúmeroDeDocumento:Documento.Estado); y 6 particiones sobre otras 2 variables, una variable de Entrada (FechaReserva) y una variable de Estado (la Fecha de las Reservas). Se podrían combinar los valores representativos de estas particiones en hasta 4x5x6 = 120 combinaciones, si se seleccionara solo un valor representativo de cada partición.

A continuación, el Cuadro 5, la especificación de particiones correspondiente:

Cuadro 5. Formato con la tabla de particiones equivalentes de las 3 variables: CódigoDeUsuario, NúmeroDeDocumento y FechaReserva.

Empresa:		Proyecto:		Nombre Clase:	Biblioteca	Hoja No.	
Analista/Desarrollador:				Auxiliar Investigación:		Fecha	/ /
Caso de Uso (si aplica):	Reservar Documento		Método:	Reservar Documento	Tipo del Valor Retornado:		Reserva / NULL

Entrada Tipo	Objeto/Variable de Estado(Valores)	Clases válidas (valores válidos)	Clases inválidas (valores inválidos)	# de Clases x Factor
CódigoDeUsuario String	Usuario: CódigoDeUsuario, Estado (Normal/ Sancionado/ Retirado)	1. Formato válido (8 dígitos (0-9) , Existe, Estado normal	2. Existe, Sancionado o Retirado, 3. No existe 4. Formato inválido (menos/más caracteres, no dígitos) incluye "en blanco" (cero caracteres).	4

NúmeroDeDocumento Integer	Documento: NúmeroDeDocumento, Disponibilidad(Disponible/No Disponible) Estado(Prestado /Reservado)	5. Numérico de 6 digitos, en el rango 000001-010000, Existe, Se puede reservar	6. Numérico en rango, Existe, No se puede reservar, 7. Numérico en rango, No Existe 8. No numérico 9. Menos dígitos (incluye "en blanco" –cero dígitos), fuera del rango.	5
FechaReserva DD/MM/AAAA DD,MM,AAAA son Integer	Fecha : Hábil (Si/No), Reserva: Fecha, Usuario, Documento	10. Formato correcto, días mes correctos, Hábil, Usuario no tiene reserva en la fecha, documento disponible	11. Formato correcto, días mes correctos, Hábil, Usuario/Documento ya tienen reserva en la fecha 12. Formato correcto, días mes correctos, No hábil 13. Formato correcto, días mes incorrectos 14. Formato incorrecto 15. En blanco	6

Si se considera un solo valor por partición, en el caso del método "ReservarDocumento", la lista de particiones y los aún no especificados casos de prueba correspondientes, tendría 120 elementos (4 x 5 x 6 = 120), con todas las posibles combinaciones de valores representativos, de las variables de entrada y de estado (los factores) es la mostrada en el Cuadro 6, a continuación. En el caso de factores con rangos de valores, no se selecciona un único valor por partición, como se señala en el siguiente numeral.

Cuadro 6. Casos de prueba con solo un valor representativo por cada partición de las 3 variables: CódigoDeUsuario, NúmeroDeDocumento y FechaReserva.

Casos de Prueba		**Factor CódigoDeUsuario**	**Factor NúmeroDeDocumento**	**Factor FechaReserva**
#	Descripción			
1	Un solo Caso Válido	1	5	10
5	Casos inválidos	1	5	≠ 10
4	¨	1	≠ 5	10
3	¨	≠ 1	5	10
20	¨	1	≠ 5	≠ 10
15	¨	≠ 1	5	≠ 10
12	¨	≠ 1	≠ 5	10
60		≠ 1	≠ 5	≠ 10
Total	119 Casos inválidos			
Total:	120 Casos			

2.2 La Técnica de los Valores Extremos

Para seleccionar los valores de prueba de cada partición se usa la técnica de los valores extremos [6, 7] en el caso de subconjuntos de valores que sean rangos, seleccionando los valores límite de los rangos. Cada valor de prueba seleccionado se numera.

Procedimiento

Establecer valores de prueba representativos para cada partición de cada variable, según:

Si la partición no es un rango y no tiene límites (los valores no son números), seleccionar un valor representativo como valor de prueba.

Si la partición es un rango o tiene límites (los valores son numéricos):

Seleccionar un valor de prueba igual al límite (uno por cada límite).

Seleccionar valores de prueba por fuera de los límites pero cercanos a ellos (si el límite es un número entero sumarle 1 si el límite es superior, y restarle uno si el límite es inferior).

Especificar los valores representativos de las particiones en el formato correspondiente.

Numerar consecutivamente los valores de prueba para cada partición en el formato.

Ejemplo 2. Selección de Valores de Prueba de 3 Particiones

Con base en las particiones del Ejemplo 1 mostrado antes, se establecen los valores de prueba, inicialmente de la variable de Entrada CódigoDeUsuario. Los valores de la variable CódigoDeUsuario son un rango de valores, [00000000-99999999] numérico pero representados con una cadena (String). Ese formato/sintaxis de puede describir formalmente (0-9) ^{8}. Son (10) ^{8}. Es decir, que hay 100.000.000 posibles valores con Formato correcto en las particiones 1 2 y 3, es decir la partición 1 con valores Válidos y las dos particiones 2 y 3 con valores No Válidos.

Se usan los Valores Límite correspondientes a los rangos de valores con Formato correcto, es decir los dos valores 00000000 y 99999999 para definir los valores representativos, resultando en 2 valores de prueba mostrados abajo, en el formato con los valores representativos para cada una de las particiones 1, 2 y 3. También se usan los Valores Límite No Válidos correspondientes a los caracteres que no son dígitos y son adyacentes a los límites de los dígitos, para establecer los

valores representativos de las cadenas de tamaño 8 pero con caracteres que no son dígitos. Hay 2 caracteres especiales No Válidos límite para los dígitos: 1) los dos puntos ":" (adyacente al cero"0"), 2) el "slash" "/" (adyacente al nueve "9"). El número de valores de prueba que resultarían combinando estos dos caracteres con dígitos, por ejemplo 1 carácter inválido y 7 dígitos es $2 \times (2)^7 = (2)^8$, un número muy grande. Este número crece si se consideran dos, tres, etc. caracteres inválidos. Dependiendo del proyecto podría ser suficiente considerar solo cadenas con 1 carácter en la última posición.

Para el caso de valores con formato inválido por el tamaño (el número de dígitos) de la cadena, se seleccionan valores No Válidos por tener 0, 7 o 9 caracteres. El valor representativo de la cadena de tamaño 0 es una cadena nula (NULL). Las cadenas de 7 y 9 caracteres se pueden dividir entre aquellas con caracteres solo dígitos, y aquellas cadenas de 7 y 9 con hasta un carácter no dígito.

Se totalizan el número de combinaciones de valores representativos de la variable CódigoDeUsuario y del estado del objeto Usuario (25), el número de valores distintos de la variable CódigoDeUsuario (21), las posibilidades de existencia/inexistencia del objeto Usuario (2) y los posibles valores de su variable Estado (3).

La variable Numero DeDocumento es de tipo Integer y comprende un rango de números Válidos [000001-010000]. Se usan estos valores límite y los adyacentes No Válidos (000000, 010001) para establecer los valores de prueba. El total de valores representativos es de 9. La variable FechaReserva está implementada como una terna de enteros, sin usar clases de la biblioteca del lenguaje (por ejemplo Date, Calendar) y no está especificada como una clase aparte por lo cual su manejo debe probarse algo más exhaustivamente, con base en los valores límite para los números de días y de meses, y los valores límite para los días hábiles de cada semana (lunes y viernes usualmente). El total de valores representativos es de 434. Pueden ser demasiados valores pero la prueba podría automatizarse utilizando la clase Calendar. A continuación, en los Cuadros 7, y 8, la especificación de los valores de prueba de las particiones de dos de las tres variables de entrada (2 formatos):

Cuadro 7. Valores de prueba con varios valores representativos por cada partición de la variable: CódigoDeUsuario.

Caso de Uso (si aplica):	Reservar Documento	Método:	Reservar Documento	Tipo del Valor Retornado:	Reserva o NULL
Variable de Entrada: Tipo	# de Particiones:	Objetos	Variables del objeto-clase: Tipo		
CódigoDeUsuario:String	4	CódigoDeUsuario : Usuario	CódigoDeUsuario: String		Estado: Enum

# Partición	# Valores	Descripción	Variables / Valores Representativos		
			VECódigoDeUsuario	:Usuario.CódigoDeUsuario	:Usuario.Estado
1	2	1. Formato válido (8 dígitos) , Existe, Estado normal	2 valores límite {00000000,99999999}	VECódigoDeUsuario	Normal
2	4	2. Existe, Sancionado o Retirado,	"	VECódigoDeUsuario	{Retirado, Sancionado}
3	2	3. No existe	"	NO Existe objeto :Usuario Tal que: Usuario.CódigoDeUsuario == VECódigoDeUsuario	
4	17	4. Formato inválido (menos/más caracteres o no dígitos) incluye "en blanco" (cero caracteres).	-Tamaño 0: Cadena vacía NULL -Cadenas de 7 formadas con los valores límite del rango de dígitos (0,9), y hasta un no dígito (":", "/"): 6 valores { 9999999, 0000000, 000000:, 9999999:, 000000/, 9999999/ } -Cadenas de 9 idem: { 99999999, 000000000, 99999999:, 000000000:, 99999999/, 000000000/ } -Cadenas de 8 con un caracter especial límite, y el resto dígitos límite (0, 9): 4 valores { 0000000:, 9999999:, 0000000/, 9999999/ }		
Total	25	Total x Variable→	21	2	3

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

Cuadro 8. Valores de prueba con varios valores representativos por cada partición de la variable: NúmeroDeDocumento.

Caso de Uso (si aplica):	Reservar Documento	Método:	Reservar Documento	Tipo del Valor Retornado:	Reserva o NULL
Variable de Entrada: Tipo	# de Particiones:	Objetos		Variables del objeto-clase: Tipo	
NúmeroDeDocumento: Integer	5	NúmeroDeDocumento: Documento		NúmeroDeDocumento: Integer	Disponibilidad: Enumeration
Se descarta la 4	4				

# Partición	# Valores	Descripción	Valores Representativos Variables		
			VENúmeroDeDocumento VE indica que es una variable de entrada.	NúmeroDeDocumento	Disponibilidad
1	2	5. Numérico de 6 digitos, en el rango 900001-910000, Existe, Se puede reservar	2 Valores límite: {900001, 910000}	VENúmeroDeDocumento	Disponible
2	2	6. Numérico en rango, Existe, No se puede reservar,	"	VENúmeroDeDocumento	No Disponible
3	2	7. Numérico en rango, No Existe	"	NO Existe objeto CódigoDeUsuario:Usuario Tal que CódigoDeUsuario:Usuario.CódigoDeUsuario == VECódigoDeUsuario	
4	0	8. No numérico			
5	3	9. Menos dígitos (incluye "en blanco" –cero dígitos), fuera del rango.	3 valores: NULL 900000, 910001,		
Total	9	Total x Variable→	3	2	2

Puede ser adecuado no considerarse el caso 8, dado el tipo de la variable (Integer) y a la comprobación de tipos que hace el compilador.

Ejemplo 3. Determinación del número de factores a combinar y el número de sus niveles

Según la especificación de las particiones y los valores de prueba de cada variable, se tendrían demasiadas combinaciones posibles. CódigoDeUsuario tiene 25 valores de prueba, NumeroDeDocumento tiene 9, y FechaReserva asumimos que tiene 434, para un total de 25 x 9 x 434 = 97.650 posibles combinaciones. Aquí es donde se puede evaluar el impacto del número de valores de prueba de las diferentes particiones, en especial del número de valores de prueba de FechaReserva. Una forma de reducir las combinaciones a probar es probar los valores de prueba de FechaReserva solo con valores de las particiones # 1 de las variables CódigoDeUsuario y NumeroDeDocumento: 2 x 2 x 434 = 1736, y probar los valores de prueba de las demás particiones de las variables CódigoDeUsuario y NumeroDeDocumento (diferentes a la # 1) solo con algunos valores de FechaReserva: 25 x 9 x 2 = 450. Aún así son muchas combinaciones en total: 2186. En este caso se ha descompuesto del problema de combinar factores en dos problemas con 3 factores: i) uno con dos factores con dos niveles y un factor de 434 niveles, y ii) otro con un factor con 25 niveles, otro de 9 y un tercer factor de 2 niveles.

2.3 La Técnica de la Matriz Ortogonal

Una vez se tienen las particiones y los valores de prueba de todas las variables involucradas en el método, se aplica la técnica de la matriz ortogonal [7] usando matrices ortogonales de fortaleza 2 para F *factores* y N *niveles*, donde el número de *factores* F es el número de variables involucradas, y N, el número de *niveles*, es el número de valores de prueba diferentes de las variables. La fortaleza 2 significa que la matriz incluye combinaciones de valores de las variables de forma que se consideren todas las interacciones entre cualquier par de variables (factores). En este caso, la técnica de la matriz ortogonal de fortaleza 2 es equivalente a la técnica de combinación de pares. La máxima fortaleza de una matriz es F. Es decir se consideran todas las interacciones entre todos los factores. En este caso, la técnica de la matriz ortogonal de fortaleza F es equivalente al diseño experimental factorial.

Ejemplo 4. La Selección/elaboración de la matriz ortogonal

Considerando la simplificación, hay que elaborar dos matrices ortogonales, una de 2 x 2 x 434 y otra de 25 x 9 x 2. Para elaborar la matriz de 2 x 2 x 434 nos sirve como guía la L20 (10x 2^2), solo que el factor de más de dos valores es el tercero. Se combinan los primeros 217 valores (1-217) de FechaReserva con los valores (1, 1) y (2, 2) de las dos otras variables, y los restantes 217 valores (218-434) de FechaReserva con los valores (1, 2) y (2, 1) de las dos otras variables. Así resultan 868 valores de prueba en lugar de los 1736 (la mitad).

Cuadro 9. Una matriz ortogonal 2 x 2 x 434 con 868 combinaciones.

Tabla 2 x 2 x 434	Factores/Variables					
Casos de Prueba	1:	CódigoDeUsuario	2:	NumeroDeDocumento	3:	FechaReserva
1		1		1		1
2		2		2		1
3		1		1		2
4		2		2		2
						…
433		1		1		217
434		2		2		217
435		1		2		218
436		2		1		218
437		1		2		219
438		2		1		219
…						…
867		1		2		434
868		2		1		434

Si el método tuviera un parámetro de entrada (un factor) e involucra a dos atributos de los objetos (3 factores en total) todos con dos posibles valores (uno válido y otro inválido- 2 niveles), la matriz correspondiente es la tabla L4 (2^3) con cuatro filas de Cuadro 10, que muestra 4 casos de prueba con una matriz ortogonal para 2 niveles y 3 factores:

Cuadro 10. Tabla/matriz ortogonal (2^3).

Tabla L4(2^3)	**Factores**		
Casos de Prueba	**Variable 1**	**Variable 2**	**Variable 3**
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Tomada de http://www.york.ac.uk/depts/maths/tables/taguchi_table.htm

El número posible de combinaciones de valores de las tres variables, es de 2^3 , es decir que hay 8 posibles combinaciones. Sin embargo la matriz ortogonal incluye solo 4 combinaciones. Esa es la idea, que el número de casos de prueba sea sustancialmente menor que el número máximo posible. Los factores pueden tener diferente número de niveles. Por ejemplo, en http://www.york.ac.uk/depts/maths/tables/taguchi_table.htm describen una tabla L50 con 50 casos de prueba, 1 factor con 2 niveles y 11 factores con 5 niveles L50 (2^1 x 5^{11}).

2.4 Especificación de Casos de Prueba

Se usa una correspondencia entre los números de los niveles de cada factor en la matriz ortogonal seleccionada o adaptada, y los valores seleccionados para las particiones de cada variable, para elaborar la tabla con los valores que deben tener las variables de entrada en todos los casos de prueba del método en cuestión. Cada fila de la tabla obtenida, describe los valores de las entradas de un caso de prueba.

Ejemplo 5. Mapear los valores de la matriz en los valores reales de las entradas

Los casos de prueba (con los valores de las variables) se obtienen reemplazando los identificadores de valores que trae la matriz (usualmente números consecutivos desde 1), por los valores de prueba de las particiones, en el mismo orden ascendente.

Cuadro 11. Los 868 casos de prueba de la matriz ortogonal 2 x 2 x 434.

Tabla 2 x 2 x 434	**Factores/Variables**					
Casos de Prueba	**1:**	**Código DeUsuario**	**2:**	**Numero De Documento**	**3:**	**Fecha Reserva**
1	1 → 00000000		1 → 900001		1 Fecha límite 1 de los días hábiles de semana 1	
2	2 → 99999999		2 → 910000		1 Fecha límite 1 de los días hábiles de semana 1	
3	1 → 00000000		1 → 900001		2 Fecha límite 2 de los días hábiles de semana 1	
					…	
433	1 → 00000000		1 → 900001		217	
434	2 → 99999999		2 → 910000		217	
435	1 → 00000000		2 → 910000		218	
436	2 → 99999999		1 → 900001		218	
…					…	
867	1 → 00000000		2 → 910000		434	
868	2 → 99999999		1 → 900001		434	

Con base en la tabla obtenida y las especificaciones del método y la clase, se especifican los casos de prueba, donde además de los valores de las variables de entrada, se indican: el objetivo, una descripción, otras condiciones de la prueba (objetos y enlaces existentes, carga de trabajo, etc.) y los resultados esperados (el valor retornado, cambios de estado esperados en el objeto, en los enlaces, etc.).

A continuación se muestra la especificación de un caso de prueba del método "AbrirCuenta" de la clase "CuentaBancaria" en el Cuadro 12.

Revista Avances en Sistemas e Informática, Vol.8 No.2, julio de 2011 - Medellín. ISSN 1657-7663

Cuadro 12. Especificación de un caso de prueba.

Empresa:	**ACME**	**Proyecto:**	**Finanzas**	**Nombre Clase:**	**CuentaBancaria**	**Hoja No.**	**1 de 2**
Analista/Desarrollador:		**Pedro Pérez**		**Lenguaje de programación:**	**Java**	**Fecha Inicio:**	/ /
Auxiliar Investigación:		**Juan Pilas**		**Tipo de Clase:**	**Datos**	**Fecha Fin:**	/ /

Nombre Método	**AbrirCuenta**	**Tipo Resultado**	**Integer**
Parámetros (Nombre y Tipo)	**Cantidad: Integer, Número: Integer, ApellidosT: String, NombresT:String**		
# /Nombre Caso de Prueba	1		
Objetivo de la Prueba			
Descripción de la Prueba	Ensayo con Valores válidos de Cantidad, Número, ApellidosTitular y NombresTitular		
Condiciones de La prueba	Cantidad >= SaldoMínimo, Número de 12 dígitos tal que 3 dígitos iniciales son código banco, 3 siguientes dígitos son código oficina y los 6 últimos no corresponden a una cuenta existente.		
Resultados Esperados	Retorna el nuevo saldo == Cantidad Existe una cuenta nueva, en Estado== abierta, Saldo == Cantidad, NumMovimientos == 1, ApellidosTitular == ApellidosT,…		
Resultados Obtenidos	Retorna el nuevo saldo == Cantidad Existe un nuevo objeto CuentaBancaria, con Estado== Null, Saldo == Cantidad, NumMovimientos == 1, ApellidosTitular == ApellidosT,… Existe un nuevo enlace entre el nuevo objeto y los objetos Cliente,…y Banco		

III. PLANIFICACIÓN DE LA EJECUCIÓN DE LOS CASOS DE PRUEBA

Para planificar la ejecución de los casos de prueba, es conveniente agruparlos con base en la similitud de sus condiciones de las pruebas para organizar sesiones de prueba de grupos de casos que requieran la misma preparación o una muy similar. Para cada sesión se deben describir explícitamente las actividades de preparación para establecer y cumplir esas condiciones de las pruebas, y para determinar, medir, los resultados obtenidos de manera que se puedan evaluar los criterios de fallo. De acuerdo con los facilitadores (personal del proyecto) y los desarrolladores se planifica la ejecución de las sesiones de prueba registrando el método, los números (o nombres) de los casos de prueba y la fecha y la hora planificadas.

Cuadro 13. Planificación de casos de prueba.

Empresa:	ACME	Proyecto:	Finanzas	Nombre Clase:	CuentaBancaria	Hoja No.	1 de 2
Analista/Desarrollador:		Pedro Pérez		Lenguaje de programación:	Java	Fecha Inicio:	16/06/209
Auxiliar Investigación:		Juan Pilas		Tipo de Clase:	Datos	Fecha Fin:	/ /

Grupo / Métodod	# /Nombre del Caso de Prueba	Fecha Planificada	Hora Planificada	Fecha Ejecución	¿Fallo? S/N	Supervisó	Observaciones
inicial	1	01/07/2009	15:00				

IV. EJECUCIÓN DE LOS CASOS DE PRUEBA

De acuerdo con lo registrado, se preparan –la anticipación de la preparación depende de los recursos que se necesiten- y ejecutan los casos de prueba, y los detalles de los resultados obtenidos se registran junto con la especificación de caso de prueba. Se registran, la fecha de ejecución, si ¿falló? S/N, quién supervisó, y si es del caso se describe el fallo o se anotan algunas observaciones.

Una vez se ejecuten los casos de prueba de cada método de las clases del proyecto, se elabora el resumen de la ejecución de los casos de prueba, donde se totaliza el número de casos planificados, ejecutados, y con fallo, y el porcentaje % de fallo (cociente del número de casos de prueba fallados y número de casos de prueba ejecutados) del método, como se muestra en el Cuadro 14.

Cuadro 14. Resumen de la ejecución de los casos de prueba de una clase.

Empresa:	**ACME**	**Proyecto:**	**Finanzas**	**Nombre Clase:**	**CuentaBancaria**	**Hoja No.**	**1 de 2**
Analista/Desarrollador:		**Pedro Pérez**		**Lenguaje de programación:**	**Java**	**Fecha Inicio:**	/ /
Auxiliar Investigación:		**Juan Pilas**		**Tipo de Clase:**	**Datos**	**Fecha Fin:**	/ /

Grupo / Método	# Casos de Prueba			% Fallo (2/1)	Supervisó	Observaciones
	Planifi-cados	Ejecutados (1)	Fallados (2)			
abrir	15	12	4	33%	José	Se descartaron 3 casos por redundantes
consignar	24	24	4	16%	“	
transferir	8	8	2	25%	“	
TOTAL	32	32	6	18%		

V. EXPERIMENTACIÓN CON EL PROCESO DE PRUEBAS

Hasta el momento se ha aplicado este proceso de pruebas de software orientado a objetos en dos proyectos de grado de Ingeniería de Sistemas en el año 2010, y a algunos métodos de clases desarrolladas por los participantes de un taller de desarrollo en el que se aplicó el proceso personal de software PSP a desarrollos reales, llevado a cabo en el año 2011. Un proyecto de grado realizó la validación del proceso de pruebas definido aquí para dos clases [9], utilizando los instrumentos (instructivos y formatos) definidos para el efecto y mencionados en la Figura 1. Otro, lo aplicó a algunas de las clases del desarrollo de software de una práctica de Ingeniería de Sistemas y a los desarrollos de dos proyectos de grado de Ingeniería de Sistemas [11]. El proceso descrito se pudo aplicar exitosamente a los casos mencionados, aunque se confirma que la adaptación de nuevas matrices ortogonales (no disponibles) es un proceso relativamente difícil para lo cual podrían ayudar algunas heurísticas. En el proyecto de grado [11] se hizo una semi-automatización del proceso de diseño de los casos de prueba, que si bien es un prototipo, muestra que es posible avanzar en esa dirección. En el proyecto de grado [9] además, se desarrolló un software para el registro de los datos del resumen por método de la planificación y ejecución de los casos de prueba y el porcentaje % de fallo de los métodos y de las clases. Este software permite además registrar las mediciones de las métricas orientadas por objetos que se requieran, en particular, las métricas seleccionadas en el proyecto de grado [12] y que están incluidas en el instrumento de medición publicado en 2008 [13].

VI. CONCLUSIONES

Aunque solo se ha podido aplicar de forma parcial el proceso a algunos de los métodos de algunas de las clases en cerca de 19 desarrollos de software, se pudo aplicar por parte de varias personas distintas al diseñador, y de forma exitosa. Es factible una semi-automatización del proceso de diseño de los casos de prueba, que use matrices ortogonales existentes y que aplique algunas heurísticas para la adaptación de nuevas matrices. Este proceso puede hacer uso de elementos auto-descriptivos del código para obtener datos sobre los parámetros de cada método, las variables de instancia y otros. El proceso puede considerar matrices ortogonales de fortaleza mayor a 2, para incluir la interacción de más de 2 factores. El proceso diseñado puede ser utilizado para la medición de la probabilidad de fallo de una clase, pero también podría emplearse para llevar a cabo pruebas de unidad en software orientado por objetos que no tenga altos requisitos acerca de su fiabilidad [3]. Para estos sistemas podrían usarse matrices ortogonales pero de fortaleza superior a 2, dependiendo de tales requisitos, u otras técnicas de pruebas como las pruebas combinatorias. En un futuro, se espera promover la aplicación de este proceso a desarrollos en proyectos de grado y en prácticas de sistemas, y si es posible llevarlo a empresas donde se desarrolle software.

AGRADECIMIENTOS

Este trabajo es financiado por el Departamento Administrativo de Ciencia Tecnología e Innovación COLCIENCIAS y el Servicio Nacional de Aprendizaje SENA

BIBLIOGRAFÍA

[13] Alba M., 2008. Instrumentos de Medición. Métricas Orientadas por Objetos: Instructivo y Formato para la medición de las métricas de programas orientados por objetos (programación orientada por objetos). Proyecto "Validación y Calibración del modelo COCOMO 2 y de Métricas Orientadas por Objetos para Pruebas y Mantenimiento de software orientado por objetos". UAM.

[14] Alba M., 2008. Instrumentos de Medición. Pruebas de unidad en programación orientada por objetos. Instructivos y Formatos para la realización de pruebas de clases de programas orientados por objetos. Proyecto "Validación y Calibración del modelo COCOMO 2 y de Métricas Orientadas por Objetos para Pruebas y Mantenimiento de software orientado por objetos". UAM.

[1] Briand L. C.; Wüst J.; Daly J. W. y Porter D. V., 2000. Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems. En: Journal of Systems and Software, Vol. 51, Num. 3, pp. 245-274.

[9] Castaño A., 2009. Aplicación para planificación y registro de resultados de pruebas de unidad para software orientado a objetos a nivel de clases y métodos. Proyecto de Grado Ingeniería de Sistemas, UAM. 89 P.

[2] Chidamber S. R. y Kemerer C. F., 1994. A Metrics Suite for Object Oriented Design. En: IEEE Transactions on Software Engineering, Vol. 20, Num 6, pp. 476-493.

[10] El-Emam K.; Benlarbi S.; Goel N. y Rai S., 1999. A Validation of Object-Oriented Metrics. Technical Report, National Research Council of Canada NRC/ERB-1062. 20 P.

[11] Fadul, C. J., 2010. Aplicación de técnicas de pruebas de unidad para software desarrollado en lenguajes de programación orientados a objetos en el Eje Cafetero, Proyecto de Grado Ingeniería de Sistemas, UAM. 122 P.

[12] Franco J. E., 2008. Evaluación de herramientas para la recolección automática de métricas orientadas a objetos, Proyecto de Grado Ingeniería de Sistemas, Universidad Autónoma de Manizales. 108 P.

[15] Hedayat A. S.; Sloane N. J. A. y Stufken J., 1999. Orthogonal Arrays: Theory and Applications. Springer-Verlag, 1999. Apéndice con Matrices ortogonales: http://www2.research.att.com/~njas/doc/OA.html

[3] Kuhn D. R.; Reilly M. J., 2002. An Investigation of the Applicability of Design of Experiments to Software Testing. En: Proc. 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02). IEEE.

[4] Lazic L. y Mastorakis N., 2008. Orthogonal array application for optimal combination of software defect detection techniques choices. En: W. Trans. on Comp. Vol. 7, Num 8, pp. 1319-1336.

[8] Lewis W., 2009. Software testing and continuous quality improvement. Tercera ed. CRC Press. 665 P.

[6] Piattini M. G., 2003. Análisis y diseño de aplicaciones informáticas de gestión. Una perspectiva de Ingeniería de software. Editorial Ra-Ma. 710 P.

[7] Pressman R., 2005. Ingeniería del Software, un enfoque práctico. Sexta edición. Editorial McGraw Hill interamericana. 640 P.

[5] Xu J.; Ho D. y Capretz, L. F., 2008. An Empirical Validation of Object-Oriented Design Metrics for Fault Prediction. En: Journal of Computer Science Vol. 4, Num. 7, pp. 571-577.

Universidad Nacional de Colombia Sede Medellín
Facultad de Minas

Escuela de Ingeniería de Sistemas

Pregrado

- Ingeniería de Sistemas e Informática.

Posgrado

- Doctorado en Ingeniería-Sistemas.
- Maestría en Ingeniería de Sistemas.
- Especialización en Sistemas con énfasis en:
  - Ingeniería de Software.
  - Investigación de Operaciones.
  - Inteligencia Artificial.
- Especialización en Mercados de Energía.

Áreas de Investigación

- Ingeniería de Software.
- Investigación de Operaciones.
- Inteligencia Artificial.

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/