Calidad de casos de prueba

Calidad y medición de sistema de información

Master en tecnologías informáticas avanzadas

Pedro Reales Mateo

Calidad de los casos de prueba

1. Introducción

La fase de pruebas es una de las más costosas del ciclo de vida software. En sentido estricto, deben realizarse pruebas de todos los artefactos generados durante la construcción de un producto software, lo que incluye las especificaciones de requisitos, casos de uso, diagramas de diversos tipos y, por supuesto, el código fuente y el resto de elementos que forman parte de la aplicación (como por ejemplo, la base de datos). Obviamente, se aplican diferentes técnicas de prueba a cada tipo de producto software.

A pesar de la gran cantidad de esfuerzo y de la importancia de las tareas de pruebas, el estándar internacional que define los procesos del ciclo de vida del software, la ISO 12207 [1], no define un proceso de Pruebas como tal, sino que aconseja, durante la ejecución de los procesos principales o de la organización, utilizar los procesos de soporte de Validación y de Verificación.

El proceso de Verificación establece la importancia de verificar cada uno de los productos que se van construyendo, bajo la asunción de que si lo que se va construyendo es todo ello correcto, también lo será el producto final. Igualmente, el proceso de Validación resalta la importancia de comprobar el cumplimiento de los objetivos de los requisitos y del sistema final, de suerte que podría construirse un Plan de pruebas de aceptación desde el momento mismo de tener los requisitos, que sería comprobado al finalizar el proyecto. Si tras la fase de requisitos viniese una segunda de diseño a alto nivel del sistema, también podría prepararse un Plan de pruebas de integración, que sería comprobado tras tener (o según se van teniendo) codificados los diferentes módulos del sistema.

Generalmente para realizar estas tareas de verificación y validación, correspondientes a las pruebas del sistema, se realizan *casos de prueba* usando frameworks específicos para esta tarea. Todos estos frameworks comparten la misma filosofía de pruebas: los casos de prueba se escriben en clases separadas que contienen los métodos de prueba que ejecutan los servicios ofrecidos por la clase bajo prueba. En su forma más simple, un objeto de la clase bajo prueba se construye en cada método de prueba; después, un conjunto de mensajes se envía a o; finalmente, el ingeniero de pruebas hace uso de las bibliotecas del framework para comprobar que los estados obtenidos de o son consistentes con los estados esperados.

Un ejemplo de caso de prueba (en este caso en Java) para probar el comportamiento de una cuenta bancaria sería el siguiente (Figura 1):

```
public void  prueba1 (){
        try{
                Cuenta c = new Cuenta();
                c.ingresar(1000);
                c.retirar(500);
                assertTrue(c.getSaldo()==500);
        }catch(Exception e){
                fail("No se debería haber producido ninguna excepción");
} }
```

Figura 1 - Caso de prueba de una clase cuenta

Se puede ver que tras crear la cuenta se ingresan 1000 unidades monetarias y posteriormente se retiran 500, comprobando después que el saldo de la cuenta es 500. Si se hubiera lanzado alguna excepción o el saldo no hubiera sido 500, se habría detectado un error en la clase bajo prueba.

Puesto la tarea de las pruebas es una de las más importantes y más costosas es importante asegurar que el proceso se realiza con una cierta calidad y ya que esta tarea de pruebas esta basada en la construcción de casos de prueba que se ejecutan para comprobar el correcto funcionamiento del sistema, es importante centrarnos en la calidad del conjunto de casos de prueba.

Como se muestra en este documento existen múltiples técnicas para medir o asegurar la calidad de un conjunto de casos de prueba, como las mediciones de cobertura que alcanzan las pruebas, la cantidad de fallos encontrados por la pruebas…, sin embargo, todas estas técnicas, que son estudiadas en las siguientes secciones, no contribuyen con sus mediciones a la evaluación de la calidad del conjunto de casos de prueba en el marco de un modelo de calidad propiamente dicho, sino que simplemente son intentos de mejorar la calidad de forma práctica.

De hecho no existe un modelo de calidad propiamente dicho para los conjuntos de casos de prueba, de manera que solo se podría aplicar el modelo presentado en la ISO 9126 [2], adaptándolo convenientemente a las características del software referente al conjunto de casos de prueba.

El documento esta estructurado de la siguiente manera, la sección 2 muestra como medir la calida de los casos de prueba de forma ideal propuesta por C. Pfallec et Al. [3], en sección 3 se estudian las diferentes medidas de cobertura para pruebas de caja blanca, en la sección 4 se estudian diferentes técnicas para tener valores de prueba con calidad que medidas de cobertura existen para los valores de prueba, en la sección 5 se estudia como medir la calidad de los conjuntos de casos de prueba mediante la capacidad de encontrar fallos medida con mutación, en la sección 6 se muestran cuatro métricas propuestas en [4] para medir la eficiencia

de un conjuntote casos de prueba. En la sección 7 se realiza una valoración del suso de las diferentes técnicas en la industria y se hace una crítica del uso que se debería dar alas mismas y por último en la sección 8 se muestra una pequeña conclusión de la calidad en el ámbito de las pruebas del software.

2. Cómo medir la calidad de los casos de prueba

En un reciente trabajo de C. Pfaller et Al. [3], los autores determinan que las técnicas que se usan normalmente para evaluar la calidad de los casos de prueba, como las medidas de cobertura, no son suficientes por si mismas ya que al usar únicamente una de estas técnicas se no se tienen en cuenta ciertos aspectos deseables del conjunto de casos de prueba, por lo que proponen ciertas características de los casos de prueba que hay que medir siempre.

Para ello definen tres aspectos fundamentales de la calidad de los casos de prueba. Lo primero es diferenciar perfectamente los objetos que se van a medir, segundo determinan diferentes dimensiones de calidad mediante los artefactos que hay disponibles en las diferentes fases del desarrollo y por último distinguen entre las medidas relativas y absolutas que se vana llevar a cavo.

Los objetos de medida

Pfaller determina tres objetos de medida a la hora de evaluar la calidad de los casos de prueba:

- Un único caso de prueba. Este objeto es importante por que la calidad de un único caso de prueba puede ser diferente de la calidad del conjunto de casos de prueba
- El conjunto completo de casos de prueba como un único elemento. Este es el elemento de medida más importante ya que es el que realmente nos interesa, y apara el cual tenemos que mejorar la calidad.
- El método con el cual se crean los casos de prueba. Es importante porque el método influye en la calidad del conjunto de casos de prueba final, aunque es muy difícil de medir.

Que hay que medir y cuando, dimensiones de calidad

Parta definir diferentes niveles de calidad hay que medir más cosas a parte de los casos de prueba como el código fuente o los documentos de requisitos. Según Pfaller, los artefactos que hay que medir, aunque podrían ser más, son:

- Especificación de requisitos.
- Código fuente del sistema.
- Resultados de las pruebas, que muestran los fallos encontrados.
- Profiles usados, que determinan el uso del sistema por un usuario especifico.

- FEMA y Análisis de riesgos.
- Informes de los fallos del sistema en producción, que muestran los errores no encontrados por la pruebas.

Con estos artefactos se definen 12 situaciones de medida posible que se muestran en la Figura 2.

Test Cases		X	X	X	X	X	X	X	X	X	X	X	X
vs.	Requirements Specification		X		X						X		
	Source Code			X	X							X	
	Test Results					X						X	X
	Usage Profile						X			X	X		
	FMEA / Risk Analysis							X		X			
	Failure Reports								X				X

Figura 2 - Posibles medidas interesantes para medir la calidad de los casos de prueba

Sin embargo el problema de esta propuesta es que no dice como medir estos elementos por lo que sería interesante dotar de un método de medición a cada una de las mediadas expuestas.

Medidas absolutas y relativa

Por último Pfaller, determina que las medidas para evaluar la calidad siempre deberían ser absolutas, sin embargo estas medidas absolutas generalmente no se pueden encontrar, por lo que hay que realizar medidas relativas, generalmente haciendo comparaciones entre diferentes conjuntos de casos de prueba de manera que podamos comparar dichos conjuntos y podamos elegir el mejor.

A pesar de lo expuesto en el trabajo de Pfaller, esto no deja de ser una propuesta muy teórica de lo que se debería hacer. En las siguientes secciones se van a revisar las diferentes técnicas que se usan en la práctica para determinar la calidad de los casos de prueba.

3. Criterios de cobertura (calidad de casos de prueba basados en pruebas de caja blanca)

Uno de los principales problemas a la hora de realizar las pruebas de un sistema es el gran número de casos de prueba necesarios para obtener una seguridad de que el sistema bajo prueba funciona correctamente, o con otras palabras, tener un conjunto de casos de prueba con la calidad suficiente para estar seguros que el programa bajo prueba funciona correctamente. Así, mediante los criterios de cobertura se definen ciertas reglas o aspectos que tiene que cumplir el conjunto de casos de prueba con respecto al programa bajo prueba, de manera que si

el conjunto de casos de prueba cumple con esas reglas aseguramos la calidad del mismo. Un criterio de cobertura se puede entender según M. Broy et al. [5] con dos significados diferentes:

- Como un criterio de adecuación, el cual sirve para determinar cuando un conjunto de casos de prueba tiene la calidad suficiente, por lo que el sistema bajo prueba que probado correctamente
- Como un criterio de selección el cual nos sirve para añadir casos de prueba nuevos a nuestro conjunto de casos de prueba y de esta manera mejorar la calidad del mismo.

Ambos significados dan la idea de que los criterios de cobertura son interesantes para medir la calidad de los conjuntos de casos de prueba. En los siguientes párrafos se muestran un estudio más detallado de los diferentes criterios de cobertura.

Dentro del ámbito de las pruebas de caja blanca, las cuales tienen en cuenta la estructura interna del programa bajo prueba para comprobar el correcto funcionamiento del mismo, los criterios de cobertura se pueden clasificar en base a los siguientes tres aspectos [5]:

Los **criterios estructurales** se usan para medir la calidad de un conjunto de casos de prueba. Esta calidad se mide en función del porcentaje alcanzado en función del criterio que se haya escogido. Como su propio nombre indica, estos criterios s definen en base a la estructura del programa bajo prueba, por ejemplo, si tenemos una maquina de estados, un criterio estructural podría definirse en función de los estados de la misma o en función de las transiciones. Estos criterios también se pueden usar para seleccionar nuevos casos de prueba que mejoren los porcentajes de cobertura alcanzados y así mejorar la calidad del conjunto de casos de prueba. Los **criterios funcionales** se definen en base a modelos funcionales del sistema como “casos de uso” o “profiles de usuarios” de manera que la calidad del conjunto de casos de uso se mejora mientras más funcionalidades se prueben. Un criterio de este tipo podría estar basado en los diferentes ataques que puede sufrir un sistema o las diferentes funcionalidades que soporta. Los **criterios estocásticos o estadísticos** que se definen en base a análisis del uso esperado del sistema de manera que los casos de uso serán de mayor calidad si prueban las funcionalidades del sistema más importantes, siendo estas las que más se vayan a usar.

Una clasificación bastante aceptada de los diferentes criterios de cobertura para las pruebas de caja blanca, es clasificar estos criterios en dos grandes grupos: criterios basados en el *flujo de control* y criterios basados en *flujo de datos* [6]. A pesar de que los criterios de cobertura se deberían basar en los tres aspectos comentados anteriormente la grana mayoría de los criterios que se clasifican en estos dos grupos, y que se van a estudiar más detenidamente a continuación, son criterios estructurales.

Criterios de cobertura basados en el flujo de control

Este tipo de criterios están basados en las expresiones lógicas introducidas en la especificación del sistema bajo prueba que determinan cuando existen saltos o bucles dentro de la implementación, en otras palabras, se basa en las condiciones de las estructuras If-THEN-ELSE y los LOOP que existe dentro de la implementación del sistema. Los criterios de cobertura de este tipo más usados son los siguientes:

El criterio de **cobertura de sentencias**, es el criterio más simple de todos, define que han de recorrerse todas las líneas o sentencias del código del programa al menos una vez.

El criterio de **cobertura de decisiones** o **cobertura de ramas** expuesto por primera vez en [7] requiere que cada posible valor (verdadero o falso) que pueda tomar cada una de las decisiones del sistema bajo prueba se de al menos una vez, sabiendo que una decisión es un conjunto de condicines relacionadas mediante operadores lógicos (and, or…) y una condición es un conjunto de expresiones algebraicas relacionadas con operadores relacionales (<, <=, >, >= …). Por ejemplo si tenemos un sistema que este compuesto por una única condición como: "IF (A and B) THEN S", donde A es una condición, B es otra condición y "A and B" es una decisión, un posible conjunto de casos de prueba que satisface este criterio estará compuesto por dos casos de prueba, uno que evaluara (A and B) a verdadero y otro que evaluara (A and B). Lo mano es que aunque recorramos todas las sentencias del programa puede ser que no se den todas la posibilidades, ya que podríamos tener B verdadero siempre y A que variase, lo que evitaría que se descubriera un posible error cuando B es falso.

El siguiente criterio es la **cobertura de condiciones**, el cual requiere que cada condición de cada decisión tome todos los posibles valores al menos una vez. Siguiendo con el ejemplo anterior, en este caso necesitaríamos dos casos de prueba, uno que evalúe A Falso y B Verdadero y otro A Verdadero, y B Falso. A pesar de que este criterio es más restrictivo que el anterior no alcanza la cobertura de decisiones, ya que en el ejemplo siempre se evaluaría a falso la decisión y nunca se ejecutaría S.

EL siguiente criterio de cobertura que soluciona los problemas de los dos anteriores, es el criterio de **cobertura de condición decisión**, el cual requiere que cada posible valor de cada condición de cada decisión del programa es producida al menos una vez y que además se produzca cada posible valor de cada decisión. Siguiendo el ejemplo anterior para este criterio nos bastaría con tener dos casos de prueba, uno que evaluase A falso y B falso y otro que evaluase A verdadero y B verdadero. De esta manera si un conjunto de casos de prueba cumple este criterio va a cumplir todos los criterios expuestos anteriormente.

Existe una variante de este criterio, el criterio de cobertura **modificada de condición decisión**, el cual requiere lo mismo, pero con el añadido de que cada condición afecte independientemente a cada decisión, es decir que los valores que se asigna a una condición

puede variar el resultado de la decisión. En este ejemplo necesitaríamos tres casos de prueba, un que evaluase las dos condiciones a verdadero de manera que ambas condiciones afectan independiente mente a la decisión, otro que evalúe A falso y B verdadero de manera que A afecta independientemente a la decisión y otro que evalúe A verdadero y B falso de manera que B afecta independientemente a la decisión.

El siguiente criterio es **cobertura de condiciones múltiples**, el cual requiere que todas y cada una de las posibles combinaciones de valores de las diferentes condiciones dentro de una decisión se den, para cada una de las decisiones. En nuestro ejemplo necesitaríamos cuatro casos de prueba, uno que nos de A verdadero y B verdadero, otro para A verdadero y B falso, otro para A falso y B verdadero y otro para A falso y B falso. De esta manera este criterio alcanza a todos los anteriores. Hay que tener en cuenta que si tenemos n condiciones en una decisión, el número de casos de prueba se eleva a 2^n, lo cual, en la práctica suele ser inviable porque se requiere mucho esfuerzo para alcanzar esta cobertura.

El último criterio de cobertura perteneciente al flujo de control, es el **criterio de cobertura de caminos**, el cual establece que se tienen que recorrer todos los caminos linealmente independientes que se encuentran en el grafo de flujo del sistema que se está probando. El número de caminos linealmente independientes se puede calcular con el número de McCabe, que se calcula contando las instrucciones que crean ramas. Con este criterio de cobertura nos aseguramos que se vana a ejecutar todas las posibles ejecuciones, teniendo en cuenta que los bucles solo se pueden ejecutar una vez en cada camino linealmente independiente.

Criterios de cobertura basados en el flujo de datos

Este tipo de criterios de cobertura se basa en el análisis de los flujos de los datos. Aquí se requiere que los casos de prueba ejecuten secuencias de instrucciones en las cuales se asignen valores a variables y posteriormente se usen esos valores. Para definir estos criterios de cobertura hay que definir previamente lo que es el **grafo de control** asociado al sistema bajo prueba. Este grafo está compuesto por una serie de nodos en los cuales se representan secuencias lineales de instrucciones y una serie de enlaces que representan trasferencias de control entre los nodos, de manera que a cada enlace se asocia una expresión booleanas que contiene la condición correspondiente a la trasferencia de control. Por ejemplo si tuviéramos un sistema como “S IF (A) THEN S2 ELSE S3” tendríamos un grafo con tres nodos S, S1 y S3 y dos enlaces “S-S1” con la condición A=verdadero y “S-S2” con la condición A= falso. Los criterios de cobertura que se existen basados en el flujo de datos son los siguientes:

El criterio de **cobertura de todas las definiciones**, se basa en que para cada variable exista al menos un caso de prueba de manera que este ejecute las instrucciones del sistema para

que se recorra un camino en el grafo de control del sistema en el cual se defina la variable y al menos se use una vez, es decir, que se cubra el código de creación de la variable y además se cubra el código de uso de esa variable al menos una vez. Este criterio de cobertura no es muy rigoroso porque podemos tener conjuntos de casos de prueba que dejen sin probar muchos usos de las variables.

Para solucionar ese problema existe el criterio de **cobertura de todos los usos**, en cual se basa en que para cada variable existan dentro del conjunto de casos de prueba, suficientes casos de prueba para que se recorran los caminos del grafo de control del sistema en los que se defina la variable y se alcancen todos los usos de dicha, es decir, que se ejecuten en el mismo caso de prueba la definición junto con los usos de la variable y en el caso de que no se puedan cubrir todos los usos con un único caso de prueba, añadir más casos reprueba que cubran siempre la definición de la variable hasta que se cubran todos los usos de dicha variable. Una especialización de este criterio propuesta en [8] define una diferenciación en tipo de uso de las variables dependiendo de si el uso se hace para establecer decisiones en el flujo de control o no. Este criterio a pesar de que cubre todos los usos de las variables tiene el problema de que existe la posibilidad de que a cada uso se pueda llegar desde varios caminos diferentes, porque puede ser que existan errores que se den solo llegando desde un camino concreto y no sean detectados por el conjunto de casos de prueba.

Para solucionar el problema presentado anteriormente surge el criterio de **cobertura de caminos definición/uso**, el cual se basa en recorrer todos los caminos posibles desde la definición hasta el uso de una variable, para todos los usos de una variable y para todas las variables. Obviamente este criterio de cobertura es muy costoso de conseguir y además existe un problema con los bucles ya que estos introducen la posibilidad de que existan caminos infinitos.

Una aproximación para resolver el problema de los bucles a la hora de conseguir la cobertura de caminos definición/uso, es el criterio propuesto en [9], el criterio de **k-tuplas requeridas**, el cual esta basado en que todos los caminos del grafo de control que ejecuten las pruebas tienen que cumplir que se ejecuten desde 1 a k iteraciones de los bucles, es decir, para cada bucle tendremos k caminos que pasan por el, de manera que cada camino, ejecuta una iteración más.

Otros criterios de cobertura

A parte de los criterios mencionados anteriormente existen otros criterios de cobertura que se suelen usar la práctica como: la **cobertura de funciones,** que se verifica cuando se han ejecutado todas las funciones del sistema bajo prueba; la **cobertura de llamadas**, que se verifica cuando se cubren todas las llamadas a funciones que existen en el programa bajo prueba

y la **cobertura de tablas**, que se verifica cuando se han hecho referencia a todos los elementos de las tablas

En un estudio realizado por M. Hutchins [10] se evalúa la efectividad de los conjuntos de casos de prueba que cumplen los criterios de cobertura basados en flujo de control y flujo de datos para encontrar fallos. En el experimento llevado a cavo, se evaluaron miles de conjuntos de casos de prueba sobre 130 programas con fallos derivados de siete programas de tamaño moderado. Los resultados del experimento determinaron que tanto los criterios basados en el flujo de control como los criterios basados en flujo de datos descubrían un ratio bastante alto de los falos introducidos en los programas. Sin embargo, usando estos criterios de cobertura por si solos no se descubrían el 100% de los fallos por que se llegó a la conclusión que son un buen complemento para las tareas de pruebas pero no son suficientes.

En definitiva, estos criterios de cobertura pueden ser útiles para medir la calida del los casos de prueba, siempre teniendo en cuenta que hablamos de pruebas de caja blanca y tenemos acceso al código del programa bajo prueba.

Cuado no hay acceso al código del sistema, todos los criterios expuestos anteriormente no se pueden usar, por lo que la calidad de las pruebas va a venir dada por los valores de de prueba. Así, para medir la calidad tendremos que fijarnos en la calida de los valores que usen los casos de prueba.

4. Calidad de los valores de prueba (Calidad de casos de prueba basados en pruebas de caja negra)

Las pruebas en las que no se tienen en cuenta la estructura interna del sistema bajo prueba se llaman pruebas de caja negra. A la hora de realizar este tipo de pruebas, es imposible probar una clase con todas sus posibilidades, ya que es impracticable probar todos los números enteros para un método que recibe un entero como parámetro, por ejemplo. Con lo cual, necesitamos criterios para elegir el conjunto de casos de prueba óptimos, de manera que podamos saber si el conjunto de casos de prueba tiene calidad o no. Un caso de prueba está bien elegido si cumple lo siguiente:

- Reduce el número de casos necesarios para que las pruebas sean de una calidad razonable. Esto implica que el caso ejecute el número máximo de posibilidades de entrada diferentes para así reducir el total de casos.
- Cubre un conjunto extenso de otros casos posibles, es decir, indica algo acerca de la ausencia o la presencia de defectos en el conjunto específico de entradas, así como de otros conjuntos similares.

En este tipo de pruebas existen algunas técnicas y criterios que nos pueden ayudar a seleccionar el conjunto de casos de prueba óptimo, las cuales se podrían adoptar para medir la calidad, aunque no han sido desarrolladas para ese fin, sino para seleccionar un conjunto de valores de prueba optimo.

Clases de equivalencia.

Esta técnica trata cada parámetro como un modelo algebraico donde unos datos son equivalentes a otros. Si logramos partir un rango excesivamente amplio de posibles valores reales a un conjunto reducido de clases de equivalencia, entonces es suficiente probar con un valor de cada clase, pues los demás datos de la misma clase son equivalentes. Con esta definición podríamos medir la calidad del conjunto de casos de prueba estudiando el número de clases de equivalencia cubiertas por el conjunto de casos de prueba en relación con la cantidad de casos de prueba que tiene el conjunto.

El problema está en identificar las clases de equivalencia, tarea para la que no existe una regla de aplicación universal, si no que solo existen una serie de recomendaciones: Hacer una identificación de las condiciones de las entradas del programa, es decir, restricciones de formato o contenido de los datos de entrada; A partir de ellas, identificar las cases de equivalencia que pueden ser de datos validos y de datos inválidos; Si un parámetro de entrada debe estar comprendido en un cierto rango, aparecen 3 clases de equivalencia: por debajo, en y por encima del rango; Si una entrada requiere un valor concreto, aparecen 3 clases de equivalencia: por debajo, en y por encima del rango; Si una entrada requiere un valor de entre los de un conjunto, aparecen 2 clases de equivalencia: en el conjunto o fuera de él; Si una entrada es booleana, hay 2 clases: si o no…

Análisis de Valores Límite (AVL).

La experiencia muestra que un buen número de errores aparecen en torno a los puntos de cambio de clase de equivalencia. Usualmente se necesitan 2 valores por frontera, uno justo abajo y otro justo encima. Con lo cual podríamos decir que esta técnica complementa a la anterior. Algunas reglas que se pueden seguir para identificar estos valores límites son: Si una condición de entrada especifica un rango de valores, se deben generar casos para los extremos del rango y casos no validos para las situaciones justo más allá de los extremos; Si la condición de entrada especifica un número finito y consecutivo de valores, hay que escribir casos para los números máximo, mínimo, uno más del máximo y uno menos del mínimo de valores. Si la entrada o salida de un programa es un conjunto ordenado, los casos se deben concentrar en el primero y en el último elemento.

Ambas técnicas han sido estudiadas por Hoffman et al. [11], donde se estudia los fundamentos matemáticos de estas dos técnicas y se llega a la conclusión de que son muy interesantes para la obtención de buenos valores de prueba.

Aunque con estas técnicas se seleccionen valores de prueba interesantes de calidad, lo cual podría adaptarse para medir la calidad de los casos de prueba mediante un criterio de cobertura de clases de equivalencia cubiertas o de clases de equivalencia/valores límite, en la literatura también existen criterios de cobertura para las pruebas de caja negra que se basan en las combinaciones de los valores de prueba que se pueden seleccionar con las técnicas expuestas anteriormente, manualmente o aleatoriamente.

Criterios de cobertura para los valores de las pruebas

Además de los criterios de cobertura de código, también pueden definirse criterios de cobertura para los valores de los parámetros de los casos de prueba en función de su "interés". Con estos criterios, viene a medirse el grado en que los diferentes valores interesantes seleccionados con una de las técnicas expuestas anteriormente o manualmente se utilizan en la batería de casos de prueba, ya que aunque seleccionemos valores de prueba de calidad para ejecutar las pruebas, las diferentes combinaciones de estos valores van a influir en la calidad final del conjunto de casos de prueba. De esta manera, usando los criterios de cobertura que se muestran a continuación podemos tener una medida de la calidad de los conjunto de casos de prueba a través de los valores de prueba. A continuación se presentan algunos criterios de cobertura para valores:

El criterio de **cobertura de cada uso** (**each-use**, o **1-wise**) es el más simple y el que menos calidad aporta al cponjunto de casos de prueba. Se satisface cuando cada valor interesante de cada parámetro se incluye, al menos, en un caso de prueba. Por ejemplo, supongamos que vamos a probar un sistema que dados la longitud de los tres lados de un triangulo, nos dice que tipo de triangulo es (Este problema conocido como el problema del triangulo ha sido usado ampliamente en la literatura como en [12]). Así, mediante las técnicas expuestas anteriormente podríamos seleccionar como buenos valores de prueba 0, 1 y 2 para cada uno de los parámetros correspondientes a cada unos de los lados del triangulo. Un conjunto de casos de prueba que satisface el criterio 1-wise estaría formado por los casos de prueba que ejecutara los valores siguientes: (0, 1, 2), (2, 0, 1) y (1, 2, 0) sobre la funcionalidad descrita. Está claro que este conjunto de casos de prueba no es de gran calidad porque quizá convendría tener casos de prueba que no contengan el cero, o casos de prueba en los que los tres lados sean iguales.

Para solucionar este problema existe otro criterio de cobertura para los valores de prueba más complejo que el anterior, el criterio de **cobertura pair-wise** (o **2-wise**). Este criterio requiere que cada posible par de valores interesantes de cualesquiera dos parámetros sea incluido en algún caso de prueba. Se trata de un criterio ampliamente utilizado y existen muchos trabajos en la literatura que han propuesto algoritmos de mejora para generar casos de prueba que cumplan este criterio [13]. En el ejemplo anterior del triangulo el conjunto de casos de prueba debería ejecutar todos los pares de valores mostrados a continuación al menos una vez:

- Lado1-Lado2: (0-0), (0-1), (0-2), (1-0), (1-1), (1-2), (2-0), (2-1), (2-2).
- Lado1-Lado2: (0-0), (0-1), (0-2), (1-0), (1-1), (1-2), (2-0), (2-1), (2-2).
- Lado2-Lado3: (0-0), (0-1), (0-2), (1-0), (1-1), (1-2), (2-0), (2-1), (2-2).

Por ejemplo, el caso de prueba que ejecuta los valores (0, 0, 0) cubriría los tres pares (0, 0) de las combinaciones anteriores. Así, para el conjunto de casos de prueba obtenga una cobertura pair-wise tendría cubrir todos los pares de valores mostrados. La calidad de los conjuntos de casos de prueba que cumplen este criterio de cobertura suele ser bastante buena, pero aun así existen otros criterios más estrictos.

El criterio de **cobertura t-wise** es una extensión del 2-wise, en la que cada combinación posible de valores interesantes de los "t" parámetros de la operación de que se trate se incluye en algún caso de prueba, en este caso ya no estamos ablando de pares de valores, sino de conjuntos de "t" valores. En el ejemplo del triangulo podríamos tener hasta una t =3.

Un caso especial del criterio de cobertura t-wise, es el criterio de **cobertura N-wise** en el cual N es el número de parámetros de la operación que se esté tratando, de manera que, N-wise se satisface cuando todas las combinaciones posibles de todos los parámetros se incluyen en casos de prueba. Este criterio es el más estricto que hay pero también es el que se asegura que la calidad del conjunto de casos de prueba es la máxima calidad posible con los valores de prueba aportados.

Otra manera que existe para medir la calidad de un conjunto de casos de prueba, a parte de las mencionadas anteriormente, es midiendo la capacidad del conjunto para encontrar fallos. Para ello se usa una técnica llamada de mutación.

5. Calidad del conjunto de casos de prueba mediante mutación

Como se cuenta en [14] en el contexto de las pruebas del software, un mutante es una copia del programa bajo prueba que se ha modificado introduciendo un único y pequeño cambio sintáctico que no impide que el programa compile (por ejemplo, cambiar un signo + por un * en una expresión aritmética) pero que supone un error con respecto al dominio del mismo. Así,

puede entenderse que el mutante es una versión defectuosa del programa original, es decir, el mutante es el programa original, pero en el que se ha introducido un fallo. De esta manera el conjunto de casos de prueba tendrá más calidad cuantos más fallos sea capaza de encontrar, o en términos de mutación, cuantos más mutantes sea capaza de matar .

El proceso de mutación consta de tres tareas o subprocesos: la generación de mutantes, la ejecución de los mutantes y el análisis de resultados.

En la tarea de **generación de mutantes** el objetivo es obtener las copias modificadas del programa bajo prueba. Generalmente esta tarea se realiza de forma automática mediante herramientas que utilizan operadores de mutación. Sobre esto se ha trabajado mucho en la literatura y se han generado muchos tipos de operadores de mutación como en [15]. Estos operadores de mutación realizan cambios al código del programa original de manera que introducen un fallo generando un mutante. Esta tarea o subproceso es la más importante porque dependiendo del tipo de mutantes y de operadores de mutación que se usen se van a obtener un conjunto de mutantes que son más difíciles de "matar" que otros, lo cual dará más calidad al conjunto de casos de prueba. Existen dos problemas importantes asociados a esta tarea: el primero es que el número de mutantes que se suele generar es muy grande, por ejemplo en un experimento de Mresa et al. [16] donde a partir de 11 programas con una media de 43,7 líneas de código se generaron 3211 mutantes, lo cual supone un gran esfuerzo no solo en este subproceso, sino también los siguientes; y por otro lado existe el problema de los mutantes equivalen, que son copias del programa en los que se ha introducido un pequeño cambio pero que no supone un error en el dominio del sistema lo que convierte al mutante en una copia del programa bajo prueba y no se puede "matar".

La segunda tarea que hay que llevar a cavo es la **ejecución de mutantes**. El objetivo de esta tarea es ejecutar el conjunto de casos de prueba contra cada uno de los mutantes generados anteriormente y contra el programa bajo prueba anotando los resultados obtenidos. El problema de esta tarea que esta relacionado con la cantidad de mutantes generados es la gran cantidad de ejecuciones que hay que llevar a cavo, ya que en el ejemplo anterior en el que teníamos 3211 mutantes y suponiendo que tuviéramos 10 casos de prueba (son muy poco casos de prueba), tendríamos que realizar 32110 ejecuciones con el consecuente consumo de tiempo y recursos.

La última tarea que hay que realizar es el **análisis de resultados**. Esta tarea es la que se encarga de comprobar cuantos mutantes ha podido "matar" el conjunto de casos de prueba analizando las salidas obtenidas en la etapa anterior.

Ya que cobertura que un determinado conjunto de casos de prueba alcanza en un programa es un indicador de la calidad de ese conjunto de casos de prueba la basándonos en que si dado un programa P formado por n líneas de código y dos test suites t1 y t2 que sirven para probar P, podemos afirmar que t1 es mejor que t2 si t1 recorre más sentencias de P que t2, podemos hacer una analogía con la mutación y determinar que del mismo modo, cuantos más

mutantes mate un test suite, mejor es dicho test suite. De hecho en [17], se dice que un test suite es adecuado para la mutación (mutation-adequate) cuando mata el 100% de los mutantes no equivalentes. Así pues, el grado de adecuación a la mutación se mide calculando el mutation-score propuesto por primera vez en [17], utilizando la siguiente fórmula.

MS(P,T) = K / (M-E) , donde

- P = programa bajo prueba
- T = conjunto de casos de prueba
- K = Número de mutantes muertos
- M = Número de mutantes generados
- E = Número de mutantes equivalentes

El objetivo de las pruebas utilizando mutación consiste, por lo tanto, en construir casos de prueba que descubran el error existente en cada mutante. Así pues, los casos de prueba tendrán una gran calidad, al ser ejecutados sobre el programa original y sobre los mutantes y la salida de éstos difiera de la salida del programa original, ya que esto significará que las instrucciones mutadas (que contienen el error) han sido alcanzadas por los casos de prueba (o, dicho con otras palabras, que los casos de prueba han descubierto los errores introducidos).

6. Calidad del conjunto de casos de prueba mediante métricas de eficiencia

Un trabajo menos difundido en la literatura es el propuesto por S. Wagner [4], en cual se proponen la cuatro métricas para medir la eficiencia de detección de de fallos de las diferentes técnicas de pruebas. Las dos primeras métricas se basan en las medidas de cobertura y en el número de fallos encontrados, y las dos últimas en el tiempo como concepto de fiabilidad.

La primera métrica es la **eficiencia de cobertura**. Esta se basa en que existe en que una relación entre la cobertura alcanzada por un conjunto de casos de prueba y la capacidad de detección de fallos de este conjunto, aunque esta relación no es clara. Para analizar la eficiencia del conjunto de casos de prueba basándose en la cobertura alcanzada Wagner propone la siguiente métrica:

$$\eta_C(L, T_n) = \frac{c(T_n)}{t(T_n)}$$

Donde L es el tipo de cobertura alcanzado, Tn es la técnica de pruebas usada, c(Tn) es la cobertura alcanzada por esa técnica y t(Tn) es el tiempo en personas y horas gastado para probar y corregir los fallos encontrados en el sistema.

La segunda métrica es la **eficiencia de cuenta de fallos**. Esta métrica se basa en la idea de que una técnica de pruebas es más eficiente que otra si encuentra un número de fallos mayor en el mismo periodo de tiempo. De esta manera, la eficiencia de cuenta de fallos está basada en el número de fallos encontrados y el esfuerzo empleado.

$$\eta_F(T_n) = \frac{m(T_n)}{t(T_n)}$$

Donde Tn es la técnica de pruebas empleada, m(Tn) es el número de fallos encontrados y t(Tn) es el tiempo en personas y horas gastado para probar y corregir los fallos encontrados en el sistema.

La siguiente métrica propuesta es algo más compleja que las anteriores, **eficiencia de la intensidad de fallos locales**. Esta medida se basa en el espacio de tiempo que existe entre dos fallos (time-between-failure), este dato es usado para analizar el comportamiento de los fallos. Así, conociendo este dato se puede calcular la intensidad de fallos dando el número de fallos por hora por ejemplo. Así, la idea que propone Wagner es medir la intensidad de fallos de un sistema antes de aplicar una técnica de pruebas y eliminar los errores encontrados y calcularla después. Para calcular esta intensidad de fallos propone hacerlo medianta una operación del sistema o medinate pruebas operacionales, es decir, pruebas manuales. De esta manera se podría obtener un decremento de la intensidad de fallos para medir la eficiencia de la técnica de pruebas usada, y en definitiva del conjunto de casos de prueba derivados con esa técnica. Para medir la intensidad de fallos se propone la siguiente formula:

$$\lambda(O) = \frac{f(O)}{l(O)}.$$

Donde f(O) es el número de fallos encontrados mediante la *O*peración del sistema y l(O) es el tiempo empleado en al operación del sistema. Con esto se deriva la siguiente formula de eficiencia del sistema:

$$\eta_L(T_n) = \frac{\lambda(O_b) - \lambda(O_a)}{\lambda(O_b) \cdot t(T_n)}.$$

Donde Tn es la técnica de pruebas empleada entre las dos mediciones de intensidad de fallos, Ob es la operación del sistema antes de aplicar la técnica de pruebas, Oa es la operación del sistema después de aplicar la técnica de pruebas y t(Tn) es el tiempo en personas y horas gastado para probar y corregir los fallos encontrados en el sistema.

La última métrica propuesta es la más compleja de todas, **eficiencia de la intensidad de fallos por modelos**. Esta métrica también se basa en la intensidad de fallos explicada anteriormente, peor con la deferencia de que también va a usar modelos de fiabilidad del sistema para calcular esa intensidad de fallos. Así, mediante la diferencia de la intensidad de fallos del sistema tras aplicar una técnica de pruebas al sistema y la intensidad de fallos calculada con el modelo de fiabilidad se calcula la eficiencia. La formula propuesta para calcular esta eficiencia es la siguiente:

$$\eta_M(T_n) = \frac{\lambda(T_{n-1}) - \lambda(T_n)}{\lambda(T_{n-1}) \cdot t(T_n)}.$$

Donde λ(Tn-1) es la intensidad de fallos un intervalo de tiempo antes de terminar de aplicar la técnica de fallos al sistema y t(Tn) es el tiempo en personas y horas gastado para probar y corregir los fallos encontrados en el sistema.

7. Crítica y valoración

Todas las técnicas comentadas en los apartados anteriores han sido desarrolladas o adaptadas en la industria y están ampliamente aceptadas para realizar las tareas de las pruebas de manera que estas tengan una calidad aceptable, aunque desgraciadamente en la práctica esto no sea así ya que las tareas de pruebas por su naturaleza, suele ser la que más recortes sufre cuando existen retrasos o hay que re-planificar un proyecto o incluso a veces a las que menos tiempo se dedica en la planificación inicial. Esto puede ser debido a que la tarea de pruebas no produce entregables para los clientes que es lo que finalmente se vende, aunque a veces, y dependiendo de los contratos los casos de prueba forman parte de los entregables.

Volviendo a las técnicas expuesta en a lo largo de este documento, en las ocasiones en las que se usan, generalmente están destinadas a la generación de casos de prueba y no a la medición de la calidad de los casos de prueba. Si bien, a lo largo de las exposición de cada una de las técnicas se expone su utilidad para medir la calidad de un conjunto de casos de prueba, en la práctica estas técnicas se usan como métodos para genera casos de prueba. Generalmente las técnicas de prueba de caja negra, las basadas en los valores de prueba, expuesta la sección 4, se suelen usar casi siempre para generar casos de prueba en vez de para medir la calidad de un conjunto de casos de prueba dado. Sin embargo, las demás técnicas expuestas, aunque también se usen para generar casos de prueba, se consideran más apropiadas para medir la calidad de un conjunto de casos de prueba.

Hecha esta valoración del uso de las diversas técnicas expuesta se plantea el problema de qué técnica usar a la hora de generar casos de prueba y medir la calidad de los mismos. En

un proceso de pruebas lo ideal es combinar pruebas de caja negra con pruebas de caja blanca, y establecer criterios de cobertura para estimar la calidad de un conjunto del conjunto de casos de prueba de manera que cuando se obtenga la calidad deseada podamos concluir el proceso de pruebas. Así, un proceso de pruebas que combine técnicas de caja negra con técnicas de caja blanca seguirá los siguientes pasos:

1) Escribir un conjunto de casos de prueba para probar un sistema

2) Ejecutar el conjunto de casos de prueba sobre el sistema como pruebas de caja negra

3) Si el conjunto de casos de prueba descubre errores en el sistema, entonces corregir el sistema y volver al paso 2

4) Ejecutar el conjuntote casos de prueba sobre el sistema como pruebas de caja blanca

5) Si el conjunto de casos de prueba no la cobertura deseada, entonces escribir nuevos casos de prueba que se añaden a conjunto, y volver al paso 2

De esta manera se puede ajustar el nivel de exigencia de calidad y el tipo de cobertura más interesante para nuestro domino. Según la definición de calidad como "adecuación al uso", no tiene porque tener más calidad un conjunto de casos de prueba que ha encontrado más errores que otro ya que puede ser que tarde mucho más tiempo y cueste mucho más dinero lo que haga inviable ese conjunto de casos de prueba. Esto es lo que ocurre cuando queremos aplicar criterios de cobertura muy estrictos, ya que para alcanzarlos tenemos que invertir mucho esfuerzo y dinero crear casos de prueba que consigan alcanzar esos criterios, lo cual puede que para una empresa de pocos recursos tenga menos calida que un conjunto de casos de prueba que alcance un criterio menos estricto.

8. Conclusiones

Como conclusiones podemos decir, que la calida de los casos de uso se tratado a lo largo de la historia de la ingeniería del software pero desde un punto de vista diferente a como se a tratado por ejemplo la calidad de los sistemas de información. En el ámbito que nos atañe esta calidad ha sido tratada para guiar el proceso de construcción desde el principio de una forma práctica, sin crear modelos de calidad y pensando en medir la calidad de los casos de prueba, de manera que se han desarrollado diferentes técnicas que aportan una visión de cómo crear casos de prueba de calidad no con el fin de medir esta, sino con le fin de probar sistemas.

9. Referencias

1. *ISO/IEC 12207, Software Life Cycle Processes*. 1995.
2. *International Standard ISO/IEC 9126. Information technology -- Software product evaluation -- Quality characteristics and guidelines for their use, International Organization for Standardization, International Electrotechnical Commission, Geneva.* 1991.
3. Pfaller, C., et al., *Multi-Dimensional Measures for Test Case Quality* http://dx.doi.org/10.1109/ICSTW.2008.28 in *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop - Volume 00* 2008 IEEE Computer Society. p. 364-368
4. Wagner, S., *Efficiency Analysis of Defect-Detection Techniques*. 2004.
5. Christophe Gaston, D.S., *Evaluation Coverage Based Testing*, in *Model-Based Testing of Reactive Systems*, Springer-Verlag, Editor. 2005: Heidelberg. p. 293-322.
6. S. A. Vilkomir, J.P.B. *Formalization of Software testing criteria using Z notation*. in *25th International Computer Software and Applications Conference*. 2001.
7. Meyer, G., *The Art of Software Testing*, ed. J.W. Sons. 1979.
8. S. Rapps, E.J.W., *Selecting software test data using data flow information.* IEEE Transactions on Software Engineering, April 1985: p. 297-300.
9. Ntafos, S.C., *A comporation of some structural testing strategies.* IEEE Transactions on Software Engineering, April 1988: p. 367-375.
10. Hutchins, M., et al., *Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria* in *Proceedings of the 16th international conference on Software engineering* 1994 IEEE Computer Society Press: Sorrento, Italy p. 191-200
11. D. Hoffman, P.S., L. White, *Boundary Values and automated component testing.* Software Testing, Verification and Reliavility, 1999. **1**: p. 3-26.
12. Aggarwal, K.K., et al., *A neural net based approach to Test Oracle* http://doi.acm.org/10.1145/986710.986725 SIGSOFT Softw. Eng. Notes 2004 **29** (3): p. 1-6
13. Maity, S. and A. Nayak, *Improved Test Generation Algorithms for Pair-Wise Testing* http://dx.doi.org/10.1109/ISSRE.2005.23 in *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering* 2005 IEEE Computer Society. p. 235-244
14. Macario Polo, M.P., Ignacio García-Rodríguez, *Decreasing the cost of mutation testing with 2-order mutants.* Software Testing, Verification and Validation, 2008.
15. Y. S. Ma, Y.R.K., J. Offutt. *Inter-class mutation operators for java*. in *ISSRE*. 2002: IEEE Computer Society.
16. E. S. Mresa, L.B., *Efficiency of Mutation Operators and Selective Mutation Strategies: An Empirical Study.* Software Testing, Verification and Reliability, 1999. **9**: p. 205-232.
17. Hamlet, R., *Testing programs with the help of a compiler.* IEEE Transactions on Software Engineering, 1977. **3**(4): p. 279-290.