CONTRIBUCIONES CIENTÍFICAS
EN HONOR DE MIRIAN ANDRÉS GÓMEZ
(Laureano Lambán, Ana Romero y Julio Rubio, editores),
Servicio de Publicaciones, Universidad de La Rioja,
Logroño, Spain, 2010.

TC2: UNA HERRAMIENTA DE APOYO PARA LAS PRUEBAS DEL SOFTWARE

JUAN JOSÉ OLARTE LARREA

Dedicado a la memoria de Mirian Andrés Gómez

Resumen. Este artículo presenta una herramienta CASE para la generación semi-automática de casos de prueba. Aunque existe un gran número de herramientas que facilitan la ejecución de los tests, es más difícil encontrar herramientas para su diseño. El diseño de los casos de prueba es la parte más crítica en el proceso de prueba del software, puesto que el éxito de la prueba depende de su adecuada elección. Por otro lado, interesa que el conjunto de casos de prueba tenga un tamaño limitado, para no ampliar innecesariamente el tiempo dedicado a la ejecución de los tests. Nuestra herramienta, que hemos llamado TC2 (Test Case CASE), trata de cubrir esta área. TC2 genera de modo semi-automático los casos de prueba utilizando la técnica de particiones de equivalencia.

Abstract. This paper presents a CASE tool for semi-automatic generation of test cases. Although there exist a large number of tools that facilitate the execution of the tests, there is a shortage of the same ones for the design of test cases. Test cases design is the most critical activity in software testing, since on his guessed right choice the success of the test depends. On the other hand, it interests that the set of test cases is kept in a contained size, to not unnecessarily increase the time dedicated to the execution of the tests. Our tool, which we have named TC2 (Test Case CASE), tries to cover this area. TC2 generates in an semi-automatic way the test cases using the equivalence partitioning technique.

1. Introducción

El software es una parte importante de multitud de dispositivos y sistemas presentes en la sociedad actual. Estos dispositivos están gobernados por software, por tanto, su correcto funcionamiento depende de la corrección de éste.

Es habitual que durante las diferentes etapas del desarrollo del software se introduzcan errores que finalmente quedan plasmados en el código. Es necesario por tanto detectar y corregir estos errores antes de entregar el producto al cliente, de lo contrario, estos errores acabarán apareciendo durante la vida de la aplicación pudiendo tener graves consecuencias. Aunque hay muchos factores que afectan a la confiabilidad del software, como un adecuado diseño y una buena gestión del proceso de desarrollo, las pruebas del software (software testing) son el primer

Key words and phrases. Automatic test case generation, software testing, test case, CASE, equivalence partitioning, code coverage.

método que se utiliza para evaluar el software producido antes de implantarse. Por tanto, es muy importante llevar a cabo un riguroso trabajo de pruebas para asegurar que el producto obtenido cumple los requisitos establecidos [1, 2, 11].

En el proceso de las pruebas del software podemos distinguir dos etapas: una inicial de diseño de casos de prueba y otra posterior de ejecución de la prueba, utilizando los casos de prueba diseñados anteriormente. Por tanto, para conseguir el éxito de la prueba es crucial un buen diseño de casos de prueba, y para ello existen diferentes enfoques y técnicas que aseguran mayores o menores niveles de cobertura. Por lo dicho en el párrafo anterior, se comprende que una parte importante del esfuerzo dedicado al desarrollo de un producto software recae sobre la fase de pruebas. Es importante disponer de herramientas que ayuden en la tarea de la realización de las pruebas con un doble objetivo. Por un lado para asegurar el grado de confianza de las mismas, y por otro, para minimizar el esfuerzo (tiempo y costes) dedicado a ellas [1, 2].

Existen diferentes niveles de pruebas: Pruebas de Aceptación, evaluan el software en lo que respecta a requisitos de usuario; Pruebas de Sistema, evaluan el software en lo que respecta al diseño arquitectónico; Pruebas de Integración, evaluan el software en lo que respecta al diseño de subsistemas; Pruebas de Unidad, evaluan el software en lo que respecta a la implementación. La herramienta que presentamos, TC2, ayuda a obtener este doble objetivo en Pruebas de Unidad.

El artículo está estructurado como sigue. En el apartado 2, "Generación automática de casos de prueba", explicamos el objetivo que persigue nuestra herramienta, el enfoque de diseño de casos de prueba que sigue y el resultado que obtiene. En el tercer apartado, "TC2", mostramos el funcionamiento de la herramienta. Terminamos el artículo con el apartado de conclusiones y trabajo futuro.

2. Generación automática de casos de prueba

El proceso de prueba del software implica la realización de una serie de tareas. En primer lugar se diseña la prueba, para ello se determinan los casos de prueba (test case), y después, cuando el código está implementado, se ejecuta la prueba. Existen en el mercado varias herramientas que automatizan la ejecución de las pruebas (JUnit, NUnit,...) [6]. Sin embargo es más difícil encontrar herramientas que ayuden en el diseño de los casos de prueba. Existen distintas técnicas para determinar los casos de prueba, las más usuales son el enfoque de caja negra (black box) [1, 2, 4, 10, 11] y el enfoque de caja blanca (white box) [1, 2, 10, 11] con una gran variedad de técnicas dentro de cada aproximación. La herramienta que proponemos sigue el enfoque de caja negra y, más específicamente, la técnica de particiones (o bloques) de equivalencia. Según éste, debe contemplarse el código objeto de la prueba como una caja negra, es decir, sin tener en cuenta su implementación, ignorando las líneas de código, utilizando únicamente su especificación (entradas y salidas y la relación entre ellas, pre y postcondiciones) para diseñar los casos de prueba.

Obviamente, la confianza total solo se conseguiría con la prueba exhaustiva, lo cual es inabordable en la mayoría de los casos. Por tanto, se trata de encontrar ciertos datos de entrada, casos de prueba, normalmente un conjunto pequeño de

ellos, que aseguren una confianza aceptable. Siguiendo la técnica de particiones de equivalencia, se consiguen los casos de prueba dividiendo el espacio completo de cada entrada en una serie de bloques, llamados particiones o bloques de equivalencia, que contienen valores para los que el programa a probar se comporta de manera idéntica. Por tanto, tomando un único dato como representante de cada bloque se tiene el mismo grado de confianza que se obtendría probándolo para todos los valores del mismo.

TC2 analiza las entradas del código a probar y determina las particiones de equivalencia, tanto de datos válidos (los que satisfacen las precondiciones) como de datos no válidos. Una vez determinadas las particiones de equivalencia, genera los casos de prueba que cubren dichas particiones (tanto particiones válidas como no válidas). El resultado producido por TC2 consiste en un conjunto de casos de prueba. Para cada uno muestra una tabla que contiene la identificación del caso (número de caso), una descripción del mismo, los datos de entrada que forman el caso de prueba, así como las particiones de equivalencia cubiertas.

3. TC2

En este apartado mostramos el funcionamiento de nuestra herramienta para la generación automática de casos de prueba.

TC2 genera automáticamente casos de prueba siguiendo la técnica de particiones (a veces llamadas clases o bloques) de equivalencia. Por tanto, necesita conocer las condiciones que deben satisfacer los datos de entrada al módulo a probar. A partir de esas condiciones determina las clases de equivalencia de datos válidos y las de datos no válidos para cada entrada. Esta información podríamos decir que es un producto interno, puesto que no forma parte en sentido estricto del material necesario para ejecutar la prueba, es decir, los casos de prueba. No obstante, puede considerarse útil como documentación del proceso de prueba, por eso, TC2 pregunta al operador si desea esa información o no.

El proceso comienza capturando la información necesaria de cada dato de entrada (condiciones o características de las entradas) del módulo a probar. Este proceso se lleva a cabo en dos pasos. Primero se solicita el tipo de entrada, dependiendo de ello, TC2 necesita un tipo de información u otro complementaria para diseñar las particiones de equivalencia respecto de ese dato de entrada. Por ejemplo, si el tipo de entrada fuese tipo rango, la herramienta solicitaría los límites inferior y superior de dicho rango. Para cada uno de los demás tipos posibles (Nº valores, Lógico, Conjunto de valores y Tratamiento diferente), TC2 mostraría una interfaz para completar la información necesaria respecto de ese dato de entrada. Una vez concluida la introducción de la información para todas las entradas al módulo, TC2 determina los bloques de equivalencia. A modo de ejemplo, si la entrada fuese de tipo Rango, y los límites de éste fuesen limInf y limSup, crearía las siguientes particiones de equivalencia:

- Una partición válida: (C1) valores dentro del intervalo limitado por limInf y limSup.

- Dos particiones no válidas: (C2) valores menores que limInf y (C3) valores mayores que limSup.

Para los demás tipos, TC2, aplica el correspondiente criterio según la técnica de particiones de equivalencia. Una vez obtenidas los bloques de equivalencia, si el usuario lo desea, mostrará su listado.

El siguiente paso para TC2 es obtener los casos de prueba que cubran las particiones de equivalencia obtenidas. Para ello, siguiendo la técnica de particiones de equivalencia, para cubrir las clases válidas, incorpora en cada caso de prueba valores correspondientes a tantas clases válidas como sea posible. Para generar casos de prueba que cubran las clases no válidas incluye una única clase no válida en cada caso de prueba, completándolo con datos de clases válidas (para evitar que un dato erróneo enmascare otros).

3.1. Criterios de cobertura. A la hora de elegir los casos de prueba para cubrir las particiones de equivalencia pueden seguirse distintas estrategias, y según cual sea ésta se obtienen diferentes niveles de cobertura. De manera más concreta, dependiendo de cómo se combinen valores de diferentes bloques, se consiguen los siguientes criterios [1, 2]:

- All Combinations Coverage (ACoC): tomar todas las combinaciones de bloques para todas las características de las entradas
- Each Choice Coverage (ECC): un valor de cada bloque debe ser usado en al menos un caso de prueba.

Estas son las dos situaciones extremas. La primera (ACoC) ofrece un mayor nivel de confianza, pero, a cambio también resultará más costosa la ejecución de la prueba, puesto que genera un número elevado de casos de prueba. La segunda (ECC), al contrario, genera pocos casos de prueba, con lo que la ejecución de la prueba será menos costosa, a cambio de obtener un menor nivel de confianza. TC2 ofrece ambas posibilidades, y será decisión del desarrollador optar por una u otra.

Obviamente hay otros niveles de cobertura intermedios, como son Pair-Wise Coverage (PWC), T-Wise Coverage (TWC), Base Choice Coverage (BCC) y Multiple Base Choices (MBCC) [1, 2] que TC2 no contempla y que quizás en una futura versión podríamos abordar.

4. Conclusiones y trabajo futuro

Si bien para la tarea de ejecución de las pruebas se dispone de un amplio abanico de herramientas CASE que lo facilitan o automatizan, hay una escasez de las mismas para el diseño de casos de prueba. Con la herramienta presentada estamos cerca de haber completado la automatización de las pruebas del software, siguiendo el enfoque de particiones de equivalencia. Desde el punto de vista de la eficiencia en el proceso de las pruebas del software, y por tanto en el proceso de desarrollo en general, las herramientas de apoyo a la ejecución de las pruebas juegan un papel importante, en tanto que ahorran esfuerzo (tiempo y recursos). Sin embargo, no aseguran que el resultado sea el adecuado, no aseguran que las pruebas que se han realizado sean correctas, o completas si se prefiere. Es obvio

que la parte crítica del proceso de pruebas es su diseño, más que su ejecución. El éxito de la prueba depende de una elección correcta de los casos de prueba. Además, si este conjunto de casos de prueba se mantiene en un tamaño reducido, también se reducirá el esfuerzo necesario en su posterior ejecución. Por todo ello nos decidimos a trabajar en el desarrollo de TC2.

En cuanto a las líneas de trabajo futuro que nos gustaría acometer destacan las siguientes:

- Llevar a cabo un caso de estudio experimental con la herramienta para probar programas reales y así disponer de una evaluación cuantitativa de TC2 que permita chequear sus resultados.
- Extender TC2 para disponer de criterios intermedios de niveles de cobertura, como son Pair-Wise Coverage (PWC), T-Wise Coverage (TWC), Base Choice Coverage (BCC) y Multiple Base Choices (MBCC).
- Ampliar TC2 para conseguir una integración con alguna herramienta de apoyo a la ejecución de pruebas, con lo que el desarrollador dispondría de ayuda para toda la fase de pruebas.
- Ampliar TC2 para incorporar técnicas de diseño de casos de prueba de caja blanca que sería de gran utilidad para probar software de estructura compleja.

Referencias

[1] P. Ammann, J. Offutt. *Introduction to Software Testing*. Cambridge University Press, 2008.

[2] P. Ammann, J. Offutt. Using formal methods to derive test frames in category-partition testing. *Proceedings of the Ninth Annual Conference on Computer Assurance. IEEE Computer Society Press.*, 69–80, 1994.

[3] V. Basili, R. Selby. *Search-based Software Test Data Generation: A Survey, in Software Testing Verication and Reliability*. Springer, 2005.

[4] B. Beizer, J. Wiley. Black Box Testing: Techniques for Functional Testing of Software and Systems. *Software IEEE* **volumen**(13), 98–, 1996.

[5] C. Beust et al. *Next Generation Java Testing. TestNG and Advanced Concepts*. Addison-Wesley, 2008.

[6] D. Bolaños et al. *Pruebas del Software y JUnit*. Pearson Educación, 2008.

[7] P. Henry. *The testing network : an integral approach to test activities in large software projects*. Springer, 2008.

[8] M. Hutcheson. *Software testing fundamentals: methods and metrics*. Wiley Publishing, 2003.

[9] P. McMinn. *Foundations of Empirical Software Engineering. Comparing the Effectiveness of Software Testing Strategies.*. Wiley Publishing, 2004.

[10] K. Mustafa, R. A. Khan. *Software Testing. Concepts and Practices*. Alfa Science International Ltd, 2007.

[11] G. J. Myers. *The art of Software Testing*. Wiley Publishing, 2004.

[12] W. Perry. *Effective methods for software testing*. John Wiley & Sons Editors, 2006.

Departamento de Matemáticas y Computación, Universidad de la Rioja, Spain
Correo electrónico: jjolarte@unirioja.es