Pruebas estructurales evolutivas con EvoTest Framework

Arthur Baars[1] , Verónica Hernández[1] , Tanja Vos[2]

[1] Instituto Tecnológico de Informática, Valencia, España
{abaars, vhernandez}@iti.upv.es
[2] Universidad Politécnica de Valencia, España
tvos@dsic.upv.es

Abstract. Las pruebas estructurales evolutivas de software han sido ampliamente investigadas pero, hasta el momento, no se han aplicado en la industria. Con el objetivo de cambiar esto el proyecto EU Evotest (2006-2009) ha investigado métodos para mejorar el testeo evolutivo y ha implementado herramientas que dan soporte a la automatización total de la generación de pruebas. Este artículo presenta un caso de estudio que ha sido realizado en un sistema industrial de Daimler para evaluar la aplicabilidad de los resultados de EvoTest en la industria.

Keywords: Pruebas de Software; Pruebas Estructurales; Pruebas de Caja Blanca; Pruebas Automáticas; Aplicación Industrial.

1 Introducción

Al contrario que en las pruebas de caja negra, donde se verifican los requisitos funcionales, las pruebas de caja blanca usan la estructura de la implementación para el diseño de los casos de prueba. El objetivo es conseguir la máxima cobertura del código con el menor esfuerzo posible y con una selección eficiente de los casos de prueba. Las pruebas de caja blanca se aplican comúnmente durante la fase de pruebas unitarias de un proyecto software.

En el contexto de pruebas de caja blanca, un caso de prueba es un vector de variables de entrada para el código que queremos probar. La ejecución del código con cada vector de entrada genera un flujo de control específico a través del código. Mediante la ejecución del conjunto de casos de prueba que consiguen una máxima cobertura del modulo de software se aumenta la posibilidad de detectar un error de software. Además, la búsqueda de la cobertura de código máxima puede también ayudar a la detección de código inalcanzable que no sólo aumenta el tamaño del código, sino que también tiene efectos perjudiciales en el mantenimiento del código y puede apuntar en el 30% de los casos a un error en este [1]. Es más, la cobertura máxima del código es un requisito en los estándares de calidad de software modernos como ISO 26262 [2] o ICE 61508 [3].

El aumento continuo en número, tamaño y complejidad del software hace que el diseño de los casos de prueba de caja blanca sea más difícil y lleve más tiempo. En un esfuerzo de automatizar el proceso, una vía de investigación se ha centrado en la utilización de algoritmos evolutivos para la generación de los casos de prueba.

En 2006 empezó el proyecto europeo de investigación EvoTest (IST- 33472), cuyo principal objetivo es encontrar soluciones al problema de las pruebas de sistemas de software complejos a través del uso de técnicas evolutivas [5]. El Evolutionary Testing Framework (ETF) que resulta de este proyecto representa el estado del arte en la automatización de las pruebas estructurales evolutivas y además es la primera herramienta de pruebas evolutivas que se ha aplicado en distintas áreas de la industria. Además de la herramienta de pruebas estructurales, que es capaz de generar casos de prueba para funciones ANSI C, el framework también incluye un componente de pruebas de caja negra.

2 Pruebas estructurales evolutivas

Para pequeñas funciones de programación que contienen pocas sentencias, la tarea de encontrar los casos de prueba que cubran todas las ramas es relativamente simple. Sin embargo, para funciones más complejas con muchas sentencias y variables de entrada tiene sentido automatizar la tarea, y una de las aproximaciones es usar los algoritmos evolutivos.

El concepto fundamental de los algoritmos evolutivos es realizar generaciones sucesivas de combinaciones de individuos cada vez mejores hasta llegar a la optimización máxima.

Inicialmente se obtiene una primera población de individuos, normalmente aleatoria. Se evalúa cada individuo de la población calculando su grado de adecuación mediante la función de adecuación. Se seleccionan pares de individuos de la población, basándose en los grados de adecuación, mediante la estrategia de selección predefinida y se combinan de alguna manera para producir nuevos individuos (existen muchas y variadas formas de hacer esa recombinación). Adicionalmente se aplica la mutación en alguno de los individuos. Se evalúa el grado de adecuación de los nuevos individuos (hijos) y se eligen los padres e hijos supervivientes que pasarán a la segunda generación. Este proceso se repite hasta que se encuentren los individuos óptimos o se dé alguna condición de parada.

En el caso de las pruebas estructurales, el objetivo será encontrar, mediante el algoritmo evolutivo, los casos de prueba que cubran la mayor parte del código [4], [6]. Estos casos de prueba consistirán en un conjunto de valores definidos para todas las variables usadas en el código que queremos probar, tanto los parámetros declarados en la cabecera de la función como las variables globales referenciadas en ella. Por lo tanto, en este caso un individuo será un caso de prueba, y el grado de adecuación indicará si el individuo ha conseguido cubrir la rama del código que estamos buscando en ese momento o, en caso contrario, cómo de cerca ha estado de cubrirla.

3 Herramienta de pruebas estructurales ETF

Como implementación del algoritmo arriba descrito, el proyecto EvoTest ha creado la herramienta de pruebas estructurales ETF. Esta ha sido creada para generar casos de prueba de módulos de software en ANSI C. Ha sido implementada usando los lenguajes de programación Java y Objective-Caml [7] y se ha creado como plugin del IDE (Entorno de Desarrollo Integrado) Eclipse para desarrolladores en C/C++. Está construida usando componentes de proyectos open-source como Guide [8], responsable del proceso evolutivo y CIL [9], que instrumenta el código C.

Después de realizar las configuraciones pertinentes e importar el modulo software que queremos probar a Eclipse, podemos empezar con la ejecución seleccionando la función C que queremos probar. Al indicar la función aparecerá una pantalla en la que podremos modificar algunas características de las variables de la función, como los limites máximo y mínimo, indicar si pueden ser nulas (en el caso de los punteros) o, directamente, desactivar las variables dándoles un valor fijo en toda la búsqueda; esto puede ser útil cuando se conoce el posible rango de valores que puede tener una variable. Esta reducción en el espacio de búsqueda puede hacer más rápida la misma.

Después de modificar las características oportunas de cada variable, el usuario podrá empezar con la generación de los casos de prueba. La herramienta implementa el criterio de cobertura de condición/decisión [10], en el que se intenta cubrir (rama verdadera y falsa) por separado cada una de las condiciones de las sentencias. Así pues, automáticamente, la herramienta instrumentará la función que queremos probar, reemplazando cada condición dentro de cada sentencia (como *if, while, switch* y *for*) en una llamada a la función de adecuación. Como se ilustra en el siguiente código, la primera sentencia *if* (línea 6) contiene dos condiciones, y la segunda (línea 8) contiene una condición. Así pues, se insertará en cada una de estas condiciones la función de adecuación (*Eval_**). A continuación, el motor evolutivo comenzará la búsqueda de los individuos utilizando Guide [8] para generar el algoritmo de búsqueda.

<table>
<tr>
<td>

```
1 short g_short;
2
3  //version original
4 int f(int l_int)
5 {
6  if((g_short==2)&&
     (l_int <0))
7     return 1;
8  else if(g_short> 3)
9     return 2;
10    else
11    return 3;
12   }
13
```

</td>
<td>

```

14 //version instrumentada
15 int f(int l_int)
16 {
17 if((Eval_Eql_I(…,g_short,2)) &&
18     (Eval_Less_I(…,l_int, 0)))
19     return 1;
20  else
        if((Eval_Grt_I(…,
               g_short, 3)))
21             return 2;
22      else
23       return 3
```

</td>
</tr>
</table>

Fig. 5. Ejemplo de instrumentación. El código de la izquierda es el original y el código de la derecha es la versión instrumentada.

La herramienta busca los casos de prueba que cubran cada una de las ramas. Para cada individuo (candidato a caso de prueba) generado por el algoritmo evolutivo, se comprueba si cubre la rama deseada en ese momento o alguna de las otras ramas del código. Si es así, ese individuo se usará como caso de prueba. También es muy probable que el individuo que cubra la rama que deseamos también cubra otras ramas. Como ejemplo, en el código de arriba cualquier individuo que cubra la rama falsa de la primera sentencia *if* también cubrirá alguna de las ramas (verdadera o falsa) de la segunda sentencia *if*. En estos casos se marcarán esas ramas también como cubiertas.

Por último se muestra al usuario el conjunto de casos de prueba que consiguen cubrir el máximo de ramas.

4 Experimentos para evaluar el ETF

Con el fin de evaluar la herramienta de pruebas estructurales ETF, los colaboradores industriales del proyecto EvoTest han seleccionado varios casos de estudio en los que han utilizado la herramienta. Aquí nos centramos en los sistemas de Daimler[4].

Los casos de estudio de Daimler fueron realizados sobre un software de sistema de frenado de emergencia, un sistema que desempaña las ventanas traseras, un controlador de motor Powertrain global, y un motor automático de parada/arranque para mejorar la eficiencia en el consumo de combustible. La mayoría del código utilizado para el caso de estudio ha sido generado a partir de modelos funcionales. En la siguiente tabla vemos las características de las funciones que se han utilizado como caso de estudio.

[4] www.daimler.com

Funciones	Líneas de código	Ramas	Nivel de anidamiento	Parámetros de entrada optimizados
Function1	406	148	13	25
Function2	864	505	12	66
Function3	453	156	10	43
Function4	235	48	5	27
Function5	175	72	9	20
Function6	583	194	8	79
Function7	2896	964	6	139
Function8	917	146	3	51

Table 3. Características de las funciones que se han escogido para el experimento.

Para poder obtener datos objetivos sobre los resultados de la herramienta se han propuesto dos hipótesis que serán evaluadas mediante variables que se medirán durante los experimentos. Estas hipótesis son:

- ETF es más **efectivo** encontrando casos de prueba cuando se aplica en sistemas industriales, comparándolo con pruebas aleatorias (RANDOM).
- ETF es más **eficiente** encontrando casos de prueba cuando se aplica en sistemas industriales, comparándolo con las pruebas aleatorias (RANDOM).

El experimento se ejecutará controlando las variables independientes y midiendo el efecto en las variables dependientes.

Las variables independientes son:

- Código C de sistemas industriales y los objetivos de las pruebas dependiendo de los criterios de cobertura seleccionados para realizarlas.
- El conjunto de parámetros evolutivos usados para definir el motor evolutivo de ETF.

Las variables dependientes son:

- Número de casos de prueba evaluados (equivalente al número de veces que el objeto de prueba ha sido ejecutado, también equivalente al número de evaluaciones de la función de adecuación). Esta variable se mide para comparar la **eficiencia**.
- Cobertura conseguida. Esta variable se mide para poder comparar la **efectividad**.

Con el fin de obtener unos datos fiables, se repetirá el experimento 30 veces para cada objeto de prueba y cada algoritmo de búsqueda (i.e. RANDOM y ETF).

Se deben tener en cuenta también algunas amenazas a la validez de los experimentos. Las principales son:

- Diferentes niveles de experiencia cuando se seleccionan los parámetros evolutivos manualmente.
- La representatividad de los casos de estudio seleccionados.

5 Resultados de los experimentos

Los resultados obtenidos se presentan en los siguientes diagramas:

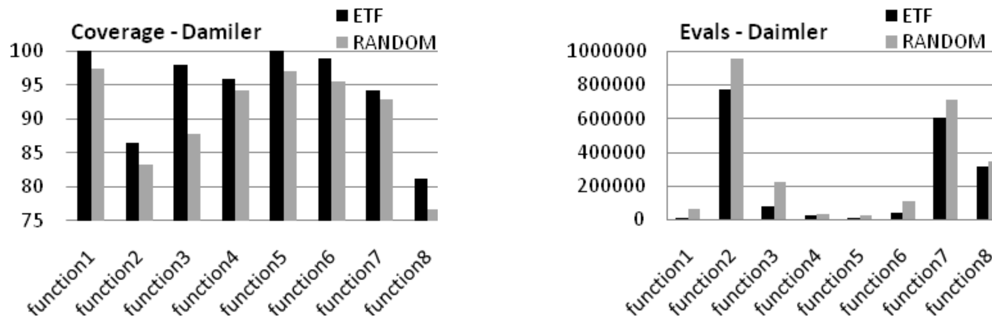

Fig. 6. Resultados obtenidos tras los experimentos.

En la parte izquierda se muestra la cobertura media conseguida por las pruebas con ETF y con RANDOM. En la parte derecha se presenta el número medio de generaciones de datos de prueba y sus ratios con ETF y RANDOM.

Para todas las funciones utilizadas la actuación de las pruebas con ETF es notablemente mejor que las pruebas con RANDOM. A pesar de que hay entre 1.1 y 6.6 veces más datos generados con las pruebas RANDOM, la cobertura conseguida no fue tan buena como con las pruebas evolutivas. Con respecto al tiempo empleado, la búsqueda en las funciones con más ramas o más niveles de anidamiento ha tardado entre 2 y 3 horas; sin embargo, en las funciones con menos ramas ha tardado un par de minutos.

6 Conclusiones

Hemos evaluado la herramienta de pruebas estructurales obtenida con el proyecto EvoTest en sistemas industriales reales y no en problemas ficticios como se ha estado haciendo hasta ahora. Los resultados son muy prometedores ya que se ha conseguido un alto nivel de cobertura que habría sido imposible alcanzar manualmente sin un esfuerzo extraordinario. Sin embargo, el prototipo ETF aun sufre algunas limitaciones que serán resueltas en versiones futuras, como la falta de soporte para punteros, variables volátiles e instrumentaciones multi-función. Este es el camino a seguir para reducir las barreras de aceptación en la industria de las pruebas evolutivas en general y del ETF en particular.

Agradecimientos

Agradecemos al equipo de Daimler su ayuda y apoyo para la redacción de este artículo. Este trabajo de investigación está apoyado por la subvención EU IST-33472 (EvoTest).

References

1. The MathWorks, Inc., "Using PolySpace Results," PolySpace Products for C User's Guide, March 2009. [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/polyspace/c_ug/index.html?/access/helpdesk/help/toolbox/polyspace/c_ug/brzsavx-1.html#brzsavx-5. [Accessed: Mar. 9, 2009].
2. ISO/CD 26262: Road Vehicles-Functional Safety, committee draft, work in progress, 2008.
3. IEC 61508-3:1998, Functional safety of electrical / electronic / programmable electronic safety-related systems, Part 3: Software requirements, 1998.

4. J. Wegener, A. Baresel, and H. Sthamer, "Evolutionary test environment for automatic structural testing," Information and Software Technology, vol.43, no.14, 2001, pp. 841-854,
5. "EvoTest – Evolutionary Testing for Complex Systems," EvoTest Project Homepage, March 2009. [Online]. Available: http://www.evotest.eu. [Accessed: Dec. 19, 2008].
6. B. Jones, H. Sthamer, and D. Eyres, "Automatic structural testing using genetic algorithms," Software Engineering J., vol. 11, no. 5, September 1996, pp. 299 – 306.
7. INRIA, "Objective CAML", About Objective CAML, May 2004. [Online]. Available: http://caml.inria.fr/ocaml/index.en.html. [Accessed: Jan. 08, 2009].
8. INRIA, "GUIDE, Crossing the chasm between theory and practice in Evolutionary Algorithms," [Online]. Available: http://guide.gforge.inria.fr. [Accessed: Dec. 22, 2008].
9. G.C. Necula, S. McPeak, S.P. Rahul, and W. Weimer, "CIL: Intermediate language and tools for analysis and transformation of C programs," Proc. 11th Intl. Conf. Compiler Construction (CC 2002), LNCS 2304, Springer, 2002, pp. 213-228.
10. J.J. Chilenski and S.P. Miller, Applicability of Modified Condition/Decision Coverage to Software Testing, Software Engineering Journal, Volume 9, Issue 5, September 1994, pp. 193-200.