Evaluación y mejora de pruebas de rendimiento utilizando mutación del software: Un enfoque evolutivo

Ana B. Sánchez[1], Pedro Delgado-Pérez[2], Inmaculada Medina-Bulo[2], Sergio Segura[1]

[1] Departamento de Ingeniería Informática, Universidad de Sevilla, España
{anabsanchez, sergiosegura@}@us.es
[2] Departamento de Ingeniería Informática, Universidad de Cádiz, España
{pedro.delgado, inmaculada.medina}@uca.es

Resumen Los defectos de rendimiento del software pueden causar una importante degradación en la experiencia de usuario y dar lugar a problemas muy costosos de detectar y resolver. Las pruebas de rendimiento persiguen detectar estos defectos y reducir su impacto. Sin embargo, no existen mecanismos para evaluar la calidad de las pruebas de rendimiento, causando en muchos casos, que estos problemas pasen desapercibidos. La prueba de mutación es una técnica para evaluar y mejorar las pruebas funcionales a través de la introducción de fallos artificiales en el programa bajo prueba. En este artículo, exploramos la aplicabilidad de la prueba de mutación junto con el empleo de un algoritmo evolutivo para buscar mutantes que simulen errores de rendimiento. Esta propuesta novedosa contribuye a mejorar la confianza en las pruebas de rendimiento al mismo tiempo que reduce el coste de la prueba de mutación.

Keywords: Prueba de mutación, problemas de rendimiento, pruebas de rendimiento, algoritmos evolutivos.

1. Introducción

La prueba de mutación tiene como objetivo evaluar y mejorar la efectividad de un conjunto de casos de prueba para detectar errores de programación. Esta técnica, que ha sido empleada en su mayoría para identificar debilidades en pruebas a nivel funcional, ha sido también extrapolada a otras actividades relacionadas con la ingeniería del software [2]. Sin embargo, la aplicación de mutaciones relacionadas con propiedades no funcionales del software como el rendimiento es actualmente muy limitada, habiéndose utilizado únicamente para tratar de optimizar el rendimiento de los programas [3]. De forma más concreta, la prueba de mutación no se ha desarrollado hasta el momento como mecanismo para simular defectos que provoquen una degradación del rendimiento del software y que, por tanto, puedan ayudar a evaluar la capacidad de las pruebas de rendimiento en la detección de fallos de este tipo. Este hecho es sorprendente teniendo en cuenta

la frecuencia de este tipo de problemas [6], el impacto que provocan y lo difícil que suele ser detectarlos y subsanarlos [4].

En esta propuesta de trabajo se plantea la apertura de una nueva línea de investigación enfocada a la aplicación de mutaciones software que nos ayuden a evaluar las pruebas diseñadas en relación al rendimiento del programa. Sabiendo que los defectos de rendimiento se definen como aquellos defectos que degradan el comportamiento y eficiencia del programa sin afectar su funcionalidad, esta línea plantea el reto de encontrar mutaciones que puedan ser de utilidad para cumplir este objetivo, ya que tradicionalmente las mutaciones se han centrado en modelar fallos a nivel funcional. Además de definir nuevos operadores de mutación adaptados al rendimiento del software, existe la posibilidad de que ciertos operadores funcionales que se han venido empleando conlleven también un empeoramiento del comportamiento del software. Este hecho no ha sido detectado hasta el momento al no haber sido analizados dichos mutantes desde una perspectiva no funcional. Es por ello que en este artículo también sentamos las bases para la implementación de un algoritmo evolutivo que, siguiendo las pautas de algoritmos anteriormente planteados en la literatura, como la prueba de mutación evolutiva [1], puedan ayudarnos a encontrar mutantes funcionalmente equivalentes que degraden el rendimiento del programa. Estos retos se presentan en mayor detalle en las siguientes secciones del artículo.

2. Prueba de Mutación Aplicada al Rendimiento

La definición de un conjunto de operadores de mutación es clave para el éxito de la prueba de mutación. Para evaluar pruebas de rendimiento es necesario determinar un conjunto de operadores que sea de utilidad para este propósito.

La primera vía para encontrar operadores de mutación relacionados con el rendimiento sería estudiar errores reales de programación, que puedan suponer una degradación del rendimiento del software, para luego crear transformaciones que modelen dichos errores. Por ejemplo, Xu et al. [5] identificaron un problema de rendimiento que se produce comúnmente cuando en un bucle se crean objetos de forma innecesaria, lo que ocasiona que el recolector de basura entre en ejecución frecuentemente para eliminar estos objetos. Inspirados por este problema de rendimiento, podríamos definir un operador de mutación que moviese la creación de un objeto de fuera a dentro de un bucle, como en el siguiente ejemplo:

```
/* Programa original */
Objeto o1 = new Objeto();
for (int i=0; i < n; i++){
    ...
}
```

```
/* Programa mutante */
for (int i=0; i < n; i++){
    Objeto o1 = new Objeto();
    ...
}
```

No obstante, este tipo de operador requeriría de un preprocesado para asegurar que se mantiene la misma funcionalidad antes y después de la mutación, que según el tipo de operador podría ser más o menos complejo. Otra opción más simple, pero que se aleja respecto del modelado de errores reales, podría ser la introducción de fallos genéricos que no impliquen este análisis previo, como

por ejemplo la inserción de retardos (*sleep()*) en diversos puntos del programa que simulen la degradación del rendimiento en cuanto al tiempo de ejecución.

La segunda vía consistiría en volver a analizar los operadores de mutación tradicionalmente empleados, pero abordándolos desde el punto de vista del rendimiento. Así, se podrían utilizar mutantes equivalentes al programa original en el plano funcional, para detectar posibles anomalías en el plano no funcional. Por ejemplo, un operador que cambia el valor de una variable por otro, podría provocar un mayor número de iteraciones de un bucle (con el consecuente aumento del tiempo de ejecución) y aun así no provocar ningún cambio a nivel funcional:

```
/* Programa original */
int n = 30;
for (int i=0; i < n; i++){
    ...
}
```

```
/* Programa mutante */
int n = 300;
for (int i=0; i < n; i++){
    ...
}
```

Dado el gran número de mutantes que se pueden llegar a generar y, teniendo en cuenta que solo estamos interesados en un subconjunto de los mismos que cumpla ciertas restricciones, en la siguiente sección planteamos el uso de un algoritmo evolutivo que nos lleve a encontrar este subconjunto de mutantes.

3. Algoritmo Evolutivo

La prueba de mutación evolutiva genera tan solo un subconjunto de los posibles mutantes mediante un algoritmo evolutivo, pero de forma que en él se seleccionen aquellos que tienen mayor probabilidad de ayudar en la mejora del conjunto de casos de prueba funcional que está bajo prueba [1]. El planteamiento propuesto en nuestra línea guarda relación con este enfoque evolutivo, pero requiere de objetivos específicos para resolver el problema de la búsqueda de mutantes con comportamiento funcional equivalente que, por el contrario, permitan evaluar la capacidad de detección de fallos de las pruebas de rendimiento.

De forma más concreta, tenemos que tener en cuenta los siguientes objetivos:

1. **Búsqueda de mutantes de funcionalidad equivalente:**
   Tomando un enfoque análogo al de la prueba de mutación evolutiva, el algoritmo evolutivo debe dar el mayor valor a aquellos mutantes que no son detectados por ningún caso de prueba, e ir reduciendo el valor que se le da al resto de los mutantes dependiendo de cuántos casos de prueba revelen la mutación que contienen. Este objetivo plantea la hipótesis de que mutantes que alteren poco o nada la funcionalidad, pueden llevarnos a encontrar otras mutaciones similares que tampoco modifiquen el comportamiento del programa.

2. **Búsqueda de mutantes que degraden el rendimiento:**
   Empleando una hipótesis similar a la del apartado anterior, este objetivo plantea que mutantes que empeoran el rendimiento también pueden llevarnos a encontrar otros mutantes de la misma índole. En este caso, el objetivo debe basarse en una medida de la propiedad o propiedades relacionadas con el rendimiento del software, como el tiempo de ejecución, consumo de memoria,

etc., las cuales podrían obtenerse mediante *profilers* [5]. Este objetivo por lo tanto otorgaría una mayor valoración a aquellos mutantes en los que se observe una mayor degradación del rendimiento.

Estamos por tanto ante un problema de búsqueda con más de un objetivo que podrá ser tratado como un problema multi-objetivo o como uno mono-objetivo en el que a cada objetivo se le asigne un peso para determinar la función de aptitud. Otra opción pasaría por modelar la equivalencia como una restricción.

4. Conclusiones

En este artículo se propone la utilización de la prueba de mutación como mecanismo para evaluar y mejorar la solidez de las pruebas de rendimiento. Como un primer paso, proponemos la generación de mutantes que simulen errores de rendimiento reales y no alteren la semántica del programa. Dando un paso más hacia la innovación, planteamos aprovechar los mutantes funcionalmente equivalentes, que siempre han sido uno de los mayores problemas en la prueba de mutación por el coste que conlleva identificarlos, para buscar degradaciones en el comportamiento del programa. Este artículo propone un algoritmo evolutivo que permita dar una segunda oportunidad a este tipo de mutantes para ayudar a evaluar la calidad de las pruebas de rendimiento. El trabajo futuro ha de ir encaminado, por tanto, a determinar cuántos de estos mutantes son realmente efectivos en este propósito.

Agradecimientos: Este trabajo está parcialmente financiado por los fondos FEDER, por los proyectos nacionales del Ministerio de Economía y Competitividad DArDOS (TIN2015-65845-C3-3-R) y BELI TIN2015-70560-R y la Red de Excelencia SEBASENET (TIN2015-71841-REDT) y el Programa de Fomento e Impulso de la actividad Investigadora de la Universidad de Cádiz.

Referencias

1. Delgado-Pérez, P., Medina-Bulo, I., Segura, S., García-Domínguez, A., Domínguez-Jiménez, J.J.: Prueba de mutación evolutiva aplicada a sistemas orientados a objetos. In: XXI Jornadas de Ingeniería del Software y Base de Datos, JISBD 2016.
2. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5), 649 –678 (Oct 2011), http://dx.doi.org/10.1109/TSE.2010.62
3. Langdon, W.B., Harman, M.: Optimizing existing software with genetic programming. IEEE Transactions on Evolutionary Computation 19(1), 118–135 (2015)
4. Segura, S., Troya, J., Durán, A., Ruiz-Cortés, A.: Performance metamorphic testing: Motivation and challenges. In: International Conference on Software Engineering: New Ideas and Emerging Results Track. pp. 7–10 (2017)
5. Xu, G., Arnold, M., Mitchell, N., Rountev, A., Sevitsky, G.: Go with the flow: Profiling copies to find runtime bloat. In: ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 419–430 (2009)
6. Zaman, S., Adams, B., Hassan, A.E.: A qualitative study on performance bugs. In: IEEE Working Conference on Mining Software Repositories. pp. 199–208 (2012)