Oráculos de prueba: Un planteamiento heurístico de apoyo a decisión

Arturo H. Torres[1], María J. Escalona[1], Manuel Mejías[1], Javier J. Gutiérrez[1]

[1] Departamento de Lenguajes y Sistemas Informáticos, Universidad de Sevilla,
Avd. Reina Mercedes sn. 41040 Sevilla, España
arturoh.torres.exts®juntadeandalucia.es
{escalona,risoto,javierj}@lsi.us.es

Resumen. Uno de los mayores desafíos hacia la consecución de las pruebas 100% automáticas está relacionado con la obtención de un eficiente oráculo de pruebas. La cuestión referente a decidir si el resultado de una prueba es aceptable o no, está relacionado estrictamente con la planificación de las pruebas, y específicamente al problema de cómo derivar los casos de prueba. Esto corresponde a lo que es denominado "oráculo", idealmente es un método que provee las salidas esperadas de cada caso de prueba dado. Este trabajo presenta un planteamiento heurístico inicial que apoya a la decisión de la aceptabilidad de una prueba en el ámbito de las aplicaciones Web.

Palabras clave: Oráculos de prueba, pruebas automáticas, pruebas de software.

1 Introducción

La creciente complejidad de las aplicaciones Web ha causado que el campo de la Ingeniería Web, definida como la aplicación sistemática, disciplinada y cuantificable de aproximaciones para el desarrollo y evolución eficiente de aplicaciones de alta calidad en la World Wide Web [8] se haya desarrollado de manera muy acelerada.

Desafortunadamente, dicha complejidad no parece estar acompañada de los mecanismos adecuados que garanticen la calidad de unos sistemas de los que cada día existe mayor dependencia a nivel social, funcional y económico.

Esta carencia de calidad ha venido generando una preocupación creciente entre la comunidad científica y empresarial involucrada en el desarrollo Web. Así pues, en los últimos años surgen varias iniciativas con el objetivo de definir marcos de referencia adecuados a estas nuevas tendencias de creación de software. Dentro de estos marcos de referencia se encuentra las pruebas de software destinados a este tipo de aplicaciones. Debido a la rapidez de crecimiento y desarrollo de las aplicaciones Web, se ha necesario e indispensable un proceso de pruebas eficiente y que tienda a ser lo más automatizado posible, consiguiendo de esta manera un ahorro de tiempo y costos.

Para asegurar la calidad del software, la técnica de pruebas es uno de los métodos más eficaces. De entre los desafíos más importantes de las pruebas de software con el objetivo de automatizar las pruebas, se encuentra el oráculo de pruebas. Es decir, la cuestión referente a decidir si el resultado de una prueba es aceptable o no, la cual está relacionada estrictamente con la planificación de las pruebas, y concretamente al problema de cómo derivar los casos de prueba.

Este desafío, el de obtener un oráculo eficiente, es un tema interesante para ser planteado y discutido. En principio, las redes neuronales son una opción válida de estudio para soportar nuestro oráculo. Es decir, las tareas de decisión que actualmente la mayoría de probadores realiza, en cuanto si el resultado de una prueba es válido o no, serían soportadas por métodos heurísticos. El objetivo de este trabajo es sólo dar un planeamiento inicial heurístico que nos permita abrir un estudio sobre la viabilidad de la utilización de técnicas de decisión heurísticas para la obtención de oráculos de prueba automáticos.

La estructura del trabajo continúa con una exposición sobre los desafíos de las pruebas de software (sección 2), útiles para ubicar la importancia de los oráculos de prueba como premisas para la obtención de pruebas 100% automáticas. Seguidamente, en la sección 3 se presenta de manera más detallada lo que se persigue con las pruebas automáticas. En la sección 4 se presenta los oráculos de prueba y sus características. Luego, en la sección 5 se presenta el planteamiento inicial heurístico de los oráculos de prueba. Finalmente, la sección 6 presenta las conclusiones y trabajos futuros.

2 Desafíos de las pruebas de software

Las pruebas de software son un término bastante amplio y abarcan una extensa gama de actividades muy diferentes, desde las pruebas realizadas por el desarrollador de una pequeña pieza de código (pruebas unitarias), hasta la validación del cliente de un gran sistema de información (pruebas de aceptación).

En todas estas fases, los casos de prueba pueden ser concebidos con objetivos muy variados, tales como validar si existen desviaciones en los requisitos del usuario, evaluar la conformidad de una especificación estándar, evaluar la robustez de las condiciones de carga, o de entradas maliciosas, medir atributos como el desempeño o usabilidad, estimar la confianza operacional, etc. Además, la actividad de las pruebas podrían ser llevadas a cabo por diversos procedimientos formales, tales como la planificación y documentación rigurosa, o como las informales y ad hoc (pruebas de exploración).

Como consecuencia de esta variedad de objetivos y ámbitos, se plantean una multiplicidad de términos para las pruebas de software, lo cual ha generado confusión y muchos problemas en la investigación sobre pruebas de software.

Para aclararlo y organizarlo en una vista unificada, presentamos la propuesta de Bertolino [2] acerca de clasificación de los problemas comunes y de los muchos significados de las pruebas de software. El primer concepto a capturar es el encontrar el denominador común, si existe, entre todas las posibles facetas de las pruebas.

Bertolino propone que el denominador común puede ser una vista muy abstracta. Dada una pieza de software (cualquiera que sea en tipología, tamaño y dominio) las pruebas siempre consisten en observar una muestra de las ejecuciones de las pruebas, y dar un veredicto sobre ellos.

A partir de esta visión general, se pueden concretar diferentes casos, distinguiendo los aspectos específicos que pueden caracterizar la muestra observada.

Una vez distinguidos los aspectos específicos que pueden caracterizar a la muestra observada, es necesario tener unas orientaciones acerca de cuál es el estado actual de las propuestas relacionadas con las pruebas de software y hacia dónde se deberían de dirigir; es decir, precisamos de un *roadmap*.

Un plan de trabajo proporciona la orientación para llegar al destino deseado, a partir del punto "tú estás aquí". El *roadmap* de la investigación de las pruebas de software está organizado de la siguiente forma:

1. El punto "tú estás aquí" consiste de los últimos logros de la investigación (tomando en cuenta que algunos de estos esfuerzos están todavía en curso).
2. El destino deseado se representa en la forma de un conjunto de sueños: se usa este término, para indicar que son metas asintóticas. Son, por definición, inalcanzables y su valor se mantiene exactamente como los polos de atracción para las futuras investigaciones.
3. En el medio están los desafíos de las actuales y futuras investigaciones de pruebas de software. Estos desafíos constituyen las orientaciones a ser seguidas, en el camino hacia los "sueños", y como tales, es la parte más importante del *roadmap*.

El *roadmap* está ilustrado en la Figura 1. Dentro de él, en el centro se sitúan las investigaciones en curso y las investigaciones emergentes, con muchos tópicos maduros (los logros) sobre la izquierda, y sobre la derecha las últimas metas (los sueños). Cuatro tiras horizontales representan las rutas de investigación identificadas hacia los "sueños":

1. Teoría de pruebas universal.
2. Pruebas basadas en modelos.
3. Pruebas 100% automáticas.
4. Ingeniería de pruebas con eficacia maximizada.

Los desafíos horizontales corresponden a las interrogantes *WHY*, *HOW*, *HOW MUCH*, *WHAT*, *WHERE*, y *WHEN* sin un orden específico.

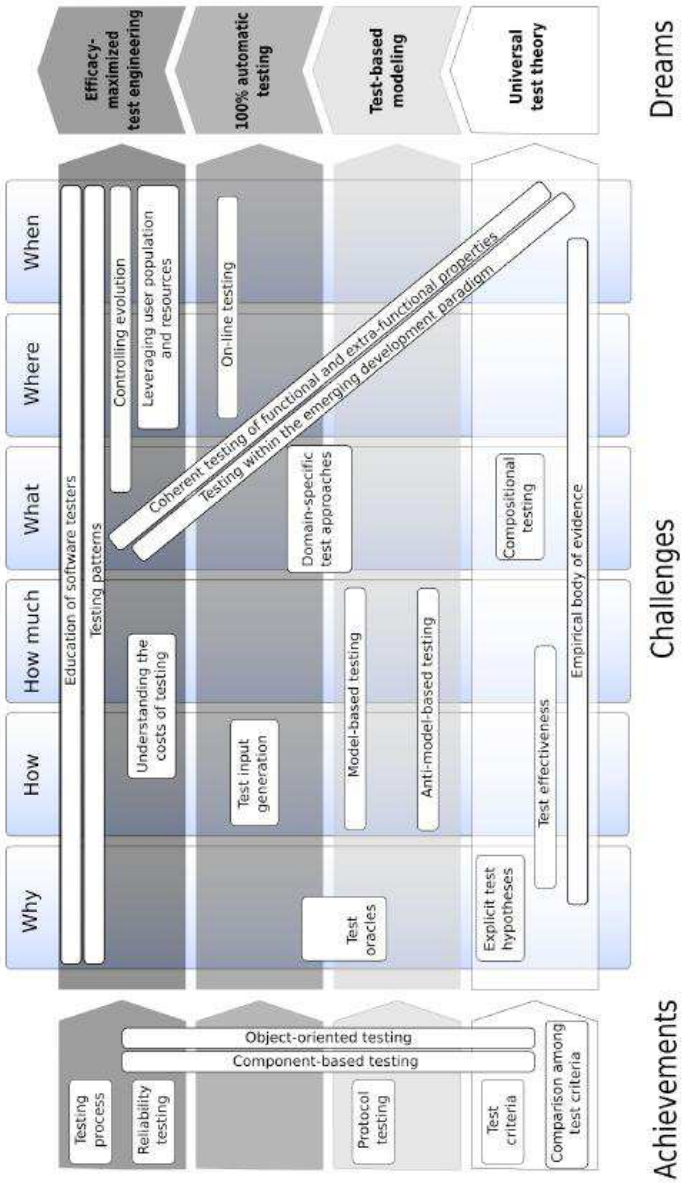

Figura. 1. Roadmap de las pruebas de software [2]

Los desafíos de la investigación en pruebas de software encuentran su lugar en este plan, verticalmente dependiendo si es un largo sueño, y hacia la que tienden principalmente, y horizontalmente de acuerdo a las cuestiones introducidas.

En base al *roadmap* detallado por Bertolino, y según la naturaleza de nuestra propuesta, consideramos oportuno centrarnos en alcanzar dos de los "sueños" mencionados: modelado basado en pruebas y las pruebas 100% automáticas.

Los siguientes apartados abordan estos conceptos, pero relacionados sólo con el sueño de las pruebas 100% automáticas, y para ello el tratamiento del desafío de los oráculos de pruebas automáticos.

3 Pruebas 100% automáticas

La automatización es una de las formas de mantener la calidad del análisis y de las pruebas, en línea con la actual complejidad del software. La investigación en ingeniería del software pone gran énfasis en la automatización de la producción del software, con una mayor parte en las herramientas de desarrollo, generando cada vez más grandes y complejas cantidades de código con menos esfuerzo.

La otra cara de la moneda es el gran peligro que tienen los métodos para evaluar calidad del software producido, en particular, los métodos de prueba que no pueden mantener el ritmo de los métodos de construcción de software.

Una gran parte de la investigación actual sobre las pruebas tiene por objeto mejorar el grado de automatización, ya sea mediante el desarrollo de técnicas avanzadas para la generación de entradas de pruebas, o para encontrar procedimientos de soporte para la automatización del proceso de prueba.

El “sueño”' vendría a ser un potente ambiente integrado de pruebas, que por sí solo, pueda, automáticamente, hacerse cargo de la generación y recuperación del código necesario (drivers, stubs, simuladores), generando los casos de prueba más adecuados, ejecutándolos y finalmente expidiendo un informe de la prueba.

Esta idea ha atraído muchos seguidores, por ejemplo, el método de pruebas continuas de Saff [13], precisamente intentan ejecutar pruebas en *background* sobre la máquina de los desarrolladores mientras ellos programan.

Se han realizado muchos pasos prometedores para las pruebas unitarias, que es ampliamente reconocida como la fase esencial que asegura la calidad del software, porque examinar unidades individuales de forma aislada puede permitir detectar tempranamente aquellos fallos sutiles que difícilmente se encuentran en el nivel de las pruebas de sistema.

Desafortunadamente, las pruebas unitarias son muchas veces mal realizadas o dejadas de lado por completo, al considerarse una actividad cara. Es así, que se necesita enfoques para hacerlo más factible dentro de los procesos de desarrollo de la industria del software.

Uno de los principales componentes que intervienen en el alto costo de las pruebas unitarias es la enorme cantidad de codificación extra, necesaria para simular el ambiente donde la unidad debe ejecutarse y en donde se ejecuta el chequeo funcional necesario para las salidas de la unidad. Para aliviar tales tareas, los *frameworks* de la familia XUnit tienen un gran éxito entre los desarrolladores. Entre ellas la más satisfactoria es JUnit [10], la cual permite la automatización del código de los casos de prueba Java.

Otro ejemplo es el proporcionado por la noción de "agitación de software" [3], una técnica de pruebas unitarias automáticas soportadas por la herramienta comercial Agitator, la cual combina diferentes análisis, tales como la ejecución simbólica, resolución de restricciones, y la generación aleatoria de entradas para la generación de los datos de entrada.

También existe otro método de pruebas unitarias, las parametrizables (PUT) [14]; es decir, las pruebas unitarias codificadas que no son fijas, sino que dependen de algunos parámetros de entrada. PUT puede describir el comportamiento abstracto en una forma concisa. Usando técnicas de ejecución simbólica y resolviendo restricciones, puede encontrar entradas para los PUTs para alcanzar un alto código de cobertura.

Los tres ejemplos citados no son ciertamente exhaustivos. La tendencia común que surge es el esfuerzo por combinar de manera eficiente los diversos tipos de análisis, y esto, junto con el aumento exponencial de los recursos computacionales disponibles, podría ser realmente la dirección hacia el sueño de la automatización 100% de las pruebas.

4 Oráculos de prueba

La cuestión referente a decidir si el resultado de una prueba es aceptable o no, está relacionado estrictamente con la planificación de las pruebas, y específicamente al problema de cómo derivar los casos de prueba. Esto corresponde a lo que es denominado "oráculo", idealmente es un método que provee las salidas esperadas de cada caso de prueba dado; de manera más realista, es una heurística que puede emitir un veredicto de pasa/fallo sobre las salidas de prueba observadas.

Aunque es evidente que una prueba de ejecución para la cual no somos capaces de discriminar entre el éxito y el fracaso, es una prueba inútil, y aunque la importancia de este problema se ha planteado muy temprano en la literatura [16], al problema del oráculo le ha prestado poca atención en la investigación y en la práctica existen pocas soluciones alternativas.

Con el incremento de la complejidad y criticidad de las aplicaciones de software, está destinado a convertirse en un obstáculo que bloquee la fiabilidad de la automatización de las pruebas.

Es más, la precisión y la eficiencia de los oráculos afectan al coste y a la eficacia de las pruebas. No se desea que las pruebas fallidas pasen desapercibidas, pero por otro lado, no queremos notificar muchos falsos positivos, los cuales echan a perder los recursos.
Se necesita encontrar métodos eficientes para la realización y automatización de las pruebas. Baresi y Young [1] proporcionan un estudio crítico de las soluciones de oráculos, concluyendo en las siguientes características:

Comportamiento y estado abstracto vs. concreto: las pruebas basadas en modelos prometen aliviar el problema de los oráculos, ya que el mismo modelo puede actuar como oráculo; sin embargo, para los oráculos basados en las descripciones abstractas del comportamiento del código, el problema sigue siendo el salvar las distancias entre las entidades concretas observadas y las entidades abstractas especificadas.

Parcialidad: convincentemente los oráculos parciales son la única solución viable para la automatización del oráculo. El reto es encontrar la mejor compensación entre precisión y costo.

Cuantificación: Para los oráculos de prueba implementados por lenguajes de especificación ejecutables, se trata de encontrar un compromiso entre la expresividad y la eficiencia. Hasta el momento no hay un claro equilibrio óptimo, ni algún método completamente satisfactorio para adaptar los cuantificadores.

Selección de casos de prueba y oráculos: Idealmente, los oráculos deben ser ortogonales a la selección de los casos de prueba; sin embargo, en las pruebas basadas por modelos, los modelos disponibles son muchas veces usados para derivar clases de pruebas y oráculos de prueba de clases específicas.

5 Planteamiento heurístico

Convencionalmente, el trabajo de las pruebas es usualmente realizado por personal experimentado con pruebas manuales. Es un duro trabajo que comúnmente cuesta más del 50% de la inversión de un proyecto [11] [12]. Hacer que este trabajo se realice sin la interferencia de personas es uno de los desafíos hacia el sueño de las pruebas automáticas. Muchas actividades de pruebas pueden ser automatizadas, generación de casos de prueba, verificación, etc. Sin embargo, los oráculos de prueba son un mecanismo o proceso utilizado para generar los resultados esperados del software bajo prueba y su automatización es un desafío actual. Para direccionar las pruebas automáticas, el método de generación automática de oráculos de prueba es el primero que tiene que tomarse en cuenta.

Ciertamente, si el personal de pruebas juzga la corrección de un programa, ellos deben saber distinguir los resultados que deberían salir. Los oráculos de prueba es ahora una rama importante para las pruebas de software automáticas, pero aún falta un entendimiento completo sobre el tema. Muchas de las investigaciones actuales acerca de la generación automática de oráculos de prueba están principalmente relacionadas con la utilización de métodos formales de verificación. Las especificaciones formales son usadas para la generación de oráculos de prueba [7] [17]. También hay trabajos relacionados con métodos dirigidos por modelos [5] [6] [4] y algoritmos heurísticos [9] [18].

Todos los métodos de prueba de software dependen de la disponibilidad de un oráculo, es decir, de algún método que verifique la exactitud del sistema bajo prueba. Un oráculo ideal proporciona un juicio infalible para la ejecución del programa. Existen varios métodos para oráculos definidos para verificar las pruebas a través de la industria del software. La captura y comparación de los resultados es fundamental para el éxito de las pruebas de software [1].

Las pruebas automáticas requieren de un trabajo más complicado con relación a las pruebas manuales. Los probadores humanos tienen la ventaja de poder decidir de acuerdo a su experiencia personal. No obstante, un probador humano al evaluar el comportamiento de un programa tiene como desventaja la exactitud limitada o el costo que ello significa. Para que sea posible una prueba automática, es condición necesaria que el oráculo de pruebas que proporciona el veredicto de pasa/fallo del caso de prueba sea también automático. La Figura 2 muestra el esquema del planteamiento propuesto, en donde una red neuronal se encargue de ser el verificador. Tanto la topología como la definición de las entradas del oráculo han de ser estudiadas, así como el tipo de entrenamiento que deba tener.

.

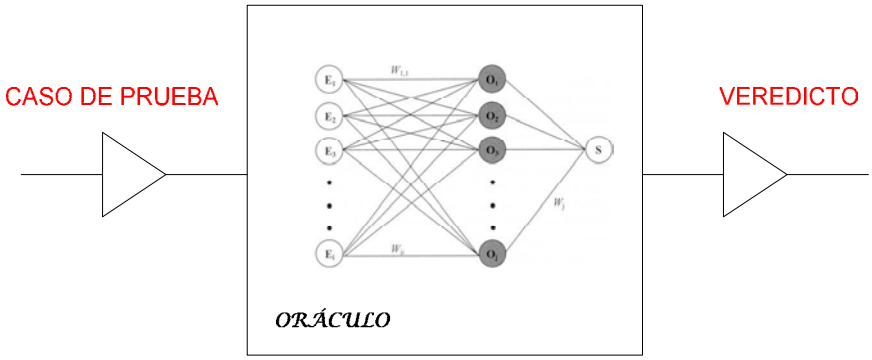

Figura. 2. Planteamiento de oráculo de prueba heurístico [2]

6 Conclusiones y trabajos futuros

A partir del estudio comparativo realizado por nuestro grupo de investigación [15], con el objetivo del estudio de las pruebas automáticas en un contexto MDWE (Model-Driven Web Engineering), en dicho estudio se identificaron diversos desafíos que deben ser abordados: automatización del proceso, enfoque MDA (Model-Driven Architecture), reglas de transformación, proceso basados en navegación y finalmente los oráculos de pruebas eficientes. Este trabajo es un planteamiento inicial sobre el cómo abordar los oráculos de pruebas con métodos heurísticos. A pesar de ser un planteamiento superficial, nuestra intención es obtener el *feedback* y estudiar la viabilidad de la utilización de técnicas de decisión heurísticas para la obtención de oráculos de prueba automáticos.

Agradecimientos. Esta trabajo ha sido apoyado por el proyecto QSimTest (TIN2007-67843-C06 03) y el proyecto RePRIS del Ministerio de Educación y Ciencia (TIN2007-30391-E), España.

Referencias

1. Baresi, L., Youngh, M.: Test oracles. Technical report, Dept. of Comp. and Information Science, Univ. of Oregon (2001).
2. Bertolino, A.: Software testing research: Achievements, challenges, dreams. In FOSE ’07: Future of Software Engineering, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society (2007).
3. Boshernitsan, M., Doong, R., Savoia, A.: From daikon to agitator: lessons and challenges in building a commercial tool for developer testing. In ISSTA ’06: Proceedings of the 2006 international symposium on Software testing and analysis, pages 169–180, New York, NY, USA. ACM Press (2006).
4. Callahan, J. R., Easterbrook, S. M.: Generating Test Oracles via Model Checking, Technique Report, NASA/WVU Software Research Lab (1998).
5. Clarke, E. M., Grumber, O., Long, D.: Model Checking and Abstraction. In Proceedings of the Nineteenth Annual ACM Symposium on Applications, North-Holland (1993).
6. Clarke, E. M., Grumberg O., Long, D.: Verification tools for finite-state concurrent system. Springer Berlin, Lecture Notes in Computer Science, Volume 803 (1994).
7. Dillon, L. K., Ramakrishna, Y. S.: Generating Oracles from your favorite Temporal Logic Specifications. Proc of the 4th ACM SIGSOFT Symp on the Foundations of Software Engineering, pp 106-117 (1996).
8. Heuser, L.: The real world or web engineering? In Proceedings of the 4th International Conference on Web Engineering. Volume 3140, page 15, Berlin, Springer (2004).
9. Hoffman, D.: Heuristic test oracles. Software Testing and Quality Engineering, Vol. 12, (1999).
10. JUnit.org. http://www.junit.org/index.htm
11. Jungmayr, S.: “Reviewing Software Artifacts for Testability”, Barcelona, Spain, Proceedings of the EuroSTAR’99, (1999).
12. Pressman, R.: Software Engineering: A Practitioner’s Approach, (2004).

13. Saff, D., D. Ernst M.: An experimental evaluation of continuous testing during development. In ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis, pages 76–85, New York, NY, USA. ACM (2004).
14. Tillmann, N., Schulte, W.: Unit tests reloaded: parameterized unit testing with symbolic execution. Software, IEEE, 23(4):38–47, (2006).
15. Torres, A. H., Escalona, M. J., Mejías, M, Gutiérrez, J.: A MDA-Based Testing:A comparative study. 4th International Conference on Software and Data Technologies, ICSOFT, Bulgary (2009).
16. Weyuker, E.J.: On testing non-testable programs. The Computer Journal, 25(4):465–470 (1982)
17. Xin, W., Ji, W., Zhi-chang, Q.: An Overiview: Temporal Specification-Based Technologies of Automatic Generation Test Oracle, Vol.28, No. 7, pp127-130 (2006).
18. Zhang, D.: Applying machine learning algorithms in software development. The Proceedings of 2000 Monterey, CA (2000).