INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES

Recibido: 05/12/2018
Aceptado: 27/01/2019

Ejecución automática de pruebas en entornos empresariales de producción de sotware

Danay Larrosa Uribazo [1] Sandra Verona Marcos [1] Perla Fernández Oliva [1] Martha Dunia Delgado Dapena [1]

[1]Facultad de Ingeniería Informática de la Universidad Tecnológica de La Habana "José Antonio Echeverría", CUJAE, Calle 114 No. 11901 e/ Ciclovía y Rotonda, Marianao, La Habana, Cuba 19390.
dlarrosa@ceis.cujae.edu.cu, sverona@ceis.cujae.edu.cu perla@ceis.cujae.edu.cu
marta@ceis.cujae.edu.cu

RESUMEN
Este trabajo presenta un conjunto de buenas prácticas para introducir en las organizaciones de desarrollo de software la ejecución automática de pruebas. Se persigue como objetivo la integración de las pruebas con el entorno de trabajo para alcanzar niveles superiores de cobertura y asistir a los desarrolladores y probadores en el diseño y ejecución de los casos de prueba.
La propuesta contempla entornos de integración continua de aplicaciones, con modelos para la generación y ejecución automática de casos de prueba. Se hace un análisis de las propuestas existentes en este ámibito, sus contribuciones y limitaciones fundamentales; como punto de partida para la presentación del modelo para la ejecución automática de pruebas de software.
El modelo Mtest.search contiene procedimientos y métodos para la generación y ejecución de casos de pruebas insertados en un entorno de integración continua dentro del propio proceso de desarrollo de aplicaciones. Esta propuesta puede ser adecuada a las condiciones específicas de cada empresa según su propia plataforma de desarrollo.
Se exponen las experiencias de aplicación del modelo en un entorno de desarrollo universitario
PALABRAS CLAVES: ejecución automática de pruebas, generación automática de casos de prueba, integración continua.

ABSTRACT
This paper presents a set of good practices for introducing automatic test execution in software development organizations. The objective is to integrate the tests with the work environment to reach higher levels of coverage and to assist the developers and testers in the design and execution of the test cases.
The proposal contemplates environments of continuous integration of applications, with models for the automatic generation and execution of test cases. An analysis is made of the existing proposals in this area, their fundamental contributions and limitations; as a starting point for the presentation of the model for the automatic execution of software tests.
The Mtest.search model contains procedures and methods for generating and executing test cases inserted in a seamless integration environment within the application development process itself. This proposal can be adapted to the specific conditions of each company according to its own development platform.
The experiences of application of the model in an environment of university development are exposed.
KEYWORDS: automatic execution of test, automatic generation of test cases, continuous integration.

INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES

Vol. 3, No. 1, (Enero 2019), 32-44 ISSN 2550-6730

1. Introducción

Con el aumento de la complejidad en el desarrollo de los sistemas informáticos, las exigencias en tiempo y calidad de los clientes y la dispersión de tecnologías y métodos empleados en los procesos de desarrollo de software actuales, las investigaciones acerca del proceso de prueba y su efectividad cobran especial significación [1]. Son múltiples las investigaciones y propuestas dirigidas a mejorar la calidad de los procesos y productos vinculados con la industria de software.

En el caso particular de las pruebas los trabajos se concentran en tres grupos de propuestas: en el primero están las dirigidas a la gestión de los procesos de pruebas, su entorno de integración continua basado en software libre para el desarrollo de aplicaciones Java, mejora y vinculación con el proceso de desarrollo [2] [3] [4] [5] [6] [7] [8] [9] [10] [11], en el segundo las enfocadas en la ejecución automática de los diferentes tipos de pruebas [12] [13] [14] y en el tercero las que se centran en asistir la actividad del diseño de las pruebas [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [9] [33] [34].

Relacionadas con el primer grupo se pueden encontrar soluciones de integración continua de aplicaciones que incorporan al proceso de desarrollo las pruebas, como un elemento indispensable para conseguir un desarrollo de aplicaciones de calidad. También se abordan soluciones de frameworks y herramientas que abordan la gestión de procesos de pruebas tercerizadas o ejecutados por equipos no comprometidos con el desarrollo.

Respecto al segundo grupo existen diferentes herramientas como JUnit [13], NUnit [14] y PHPUnit [12] que permiten ejecutar pruebas unitarias de forma automática, pero carecen de funcionalidades que asistan al desarrollador en el diseño de los casos de pruebas. Ello se debe a que, a pesar de que estas herramientas crean automáticamente una clase de prueba con un método de prueba vacío, el desarrollador debe llenar el método de prueba y crear el resto de los métodos que necesite. Ante esta problemática los programadores optan por hacer pruebas unitarias de forma empírica, sin tener en cuenta todos los posibles caminos de prueba.

Por último, en el caso del diseño de las pruebas, las propuestas van desde la utilización de algoritmos de optimización e inteligencia artificial para resolver el problema de la explosión combinatoria en la generación de casos de pruebas, hasta propuestas de frameworks para la automatización de algunos elementos del proceso. En este último caso las propuestas son prototipos para validar la solución teórica, pero no se han incorporado a las soluciones comerciales los elementos de generación de los casos de prueba, de forma tal que puedan ser utilizados por desarrolladores y equipos de probadores, reduciendo así el esfuerzo vinculado con esta actividad de diseño que es altamente costosa. Para abordar soluciones de este tipo es necesario integrar los tres grupos de trabajos que aparecen en la bibliografía en entornos de trabajo, es decir hacer propuestas a las empresas que integren los entornos de integración continua con herramientas de generación de casos de pruebas y herramientas de ejecución automática de pruebas, todo encaminado a detectar defectos de forma temprana.

Este artículo presenta una propuesta para hacer posible esta integración a través de un conjunto de buenas prácticas que pueden permitir a la empresa incorporar a su proceso de desarrollo técnicas y herramientas de integración continua soportadas en generación y ejecución automáticas de pruebas.

2. Materiales y Métodos

Un entorno de ejecución automática de pruebas tiene tres componentes: el entorno de integración continua de aplicaciones, las herramientas de generación automática de casos de prueba y las herramientas de ejecución automática de pruebas.

Un entorno de Integración Continua (IC) es la integración de varias herramientas, donde cada una cumple un papel importante en algunas etapas del desarrollo de software (Prueba, Implementación, Despliegue) con un enfoque en esta práctica. Actualmente se pueden encontrar muchas herramientas para desplegar un entorno de IC, cada una con sus particularidades, pero principalmente se dividen en: software propietario o privado y software libre o de código abierto [35].

Un entorno de IC debe poseer un servidor correspondiente a un Sistema de Control de Versiones (SCV) para controlar las modificaciones en el proyecto en todo momento y asegurar copias constantes del desarrollo con el objetivo de retroceder a la última copia de seguridad en caso de cometer un error grave en el software. Además, con un control de versiones, el servidor de IC se informa de cuándo ha de actuar si detecta algún cambio en el proyecto. Para usar IC, aunque no es imprescindible utilizar un servidor de IC, sí es muy recomendable, debido a que es el encargado de automatizar las tareas del proyecto entre otras funcionalidades [36].
Los entornos de IC se configuran y despliegan de acuerdo a las necesidades de cada grupo de desarrollo y sus objetivos. En el mundo pueden encontrarse variedad de ellos, pero el problema radica en que se hace difícil y complejo encontrar o realizar un procedimiento estándar para configurar y seleccionar las herramientas convenientes para los grupos de desarrollo. En la Figura 1 se muestra el ciclo del proceso de IC.

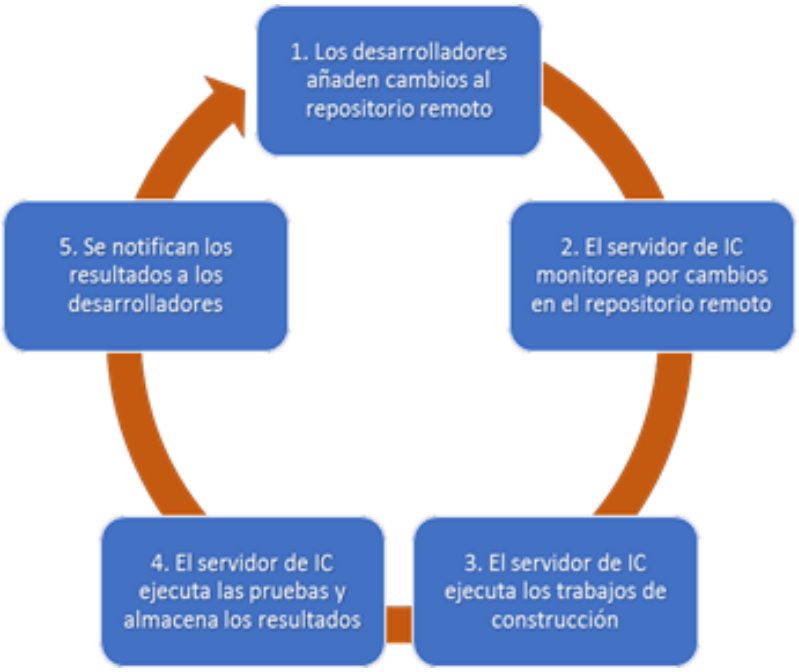

Figura 1: Proceso de Integración Continua.

El tercer componente relacionado con las herramientas de ejecución automática de pruebas está también condicionado por el entorno de desarrollo de la empresa. En el mercado están disponibles herramientas comerciales con este fin, algunas de ellas propietarias y otras no lo son, como, por ejemplo: JUnit, NUnit y PHPUnit. Es necesario que la empresa evalúe cuál de ellas se ajusta mejor a su presupuesto y políticas de negocio.
El segundo componente requiere una evaluación de las soluciones existentes para la generación automática de casos de pruebas, teniendo en cuenta que la selección debe considerar la integración con el resto de los componentes del modelo. En este artículo se presenta una propuesta específica, el modelo MTest.search.
Mtest.search consta de Flujos de Trabajo para la generación de pruebas tempranas en el entorno de producción, Modelos de Optimización para reducción de casos de pruebas funcionales y unitarias, y Herramientas Automatizadas Integradas que le dan soporte. Las recomendaciones del modelo son:

1. Contar con una estructura en el proyecto con al menos dos personas que desempeñen las funciones del ingeniero de prueba que de conjunto con clientes y/o desarrolladores realicen el diseño de

INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES
Vol. 3, No. 1, (Enero 2019), 32-44 ISSN 2550-6730

las pruebas y definan cuáles son los perfiles de prueba para cada proyecto. Además, quienes desempeñen este rol deben encargarse de mantener la implementación de las pruebas automáticas si fuera necesario.

2. Insertar las herramientas de generación y ejecución automática de pruebas en el entorno de integración continua de forma que se garantice la calidad de las nuevas soluciones implementadas antes de integrarlas con la solución ya existente.
3. Establecer relación entre el funcionamiento de las herramientas comerciales de ejecución de pruebas (aplicación cliente) y las herramientas para la generación de los casos de prueba (componente). En este sentido se propone dar tratamiento diferenciado a las pruebas unitarias y a las pruebas funcionales, teniendo en cuenta su propia naturaleza.

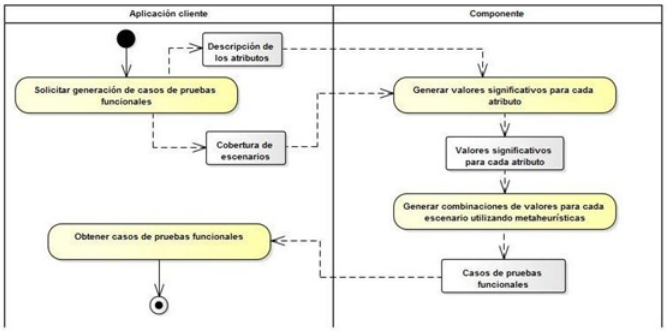

Figura 2: Secuencia de actividades para la generación de casos de pruebas funcionales y su inserción en el entorno productivo.

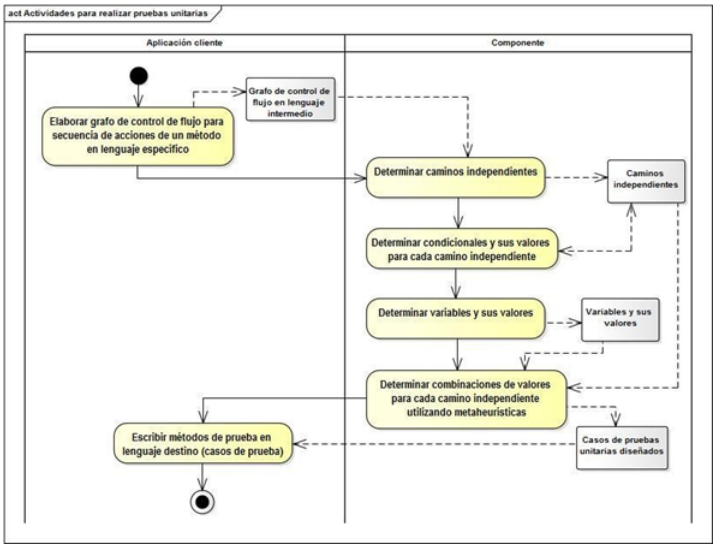

Figura 3: Secuencia de actividades para la generación de casos de pruebas unitarias y su inserción en el entorno productivo.

Para las pruebas funcionales es necesario desarrollar aplicaciones clientes que se integren con el entorno de gestión del proyecto y de gestión de pruebas funcionales que utiliza la empresa. Estas aplicaciones deben obtener la información de la funcionalidad que se pretende probar y una vez generados los escenarios y sus casos de prueba los inserten en el entorno de pruebas funcionales.

Para el caso de las pruebas unitarias, deben implementarse aplicaciones clientes, al estilo de plug-in insertados en los IDE de desarrollo, que obtengan el grafo de control de flujo (GCF) a partir del código en lenguaje fuente y que después de generados los casos de prueba (CP) los transformen en código fuente de la herramienta de ejecución automática de pruebas unitarias seleccionada.

En las Figuras 2 y 3 se muestran las actividades que han sido implementadas en el componente y las actividades que deben ser implementadas por las aplicaciones clientes en cada caso.

3. Resultados y Discusión

Las herramientas utilizadas en el entorno de IC propuesto fueron como Herramienta de Control de versiones o versionado Git y como Servidor de Integración Continua Jenkins. Se utilizó el IDE Netbeans en el entorno propuesto porque puede ser integrado fácilmente a servidores de IC como por ejemplo Hudson y Jenkins.
La Figura 4 muestra cómo se añade una instancia de un servidor Hudson - que funciona igual para Jenkins - para ser utilizado desde el IDE sin necesidad de obtener los reportes desde un navegador web.

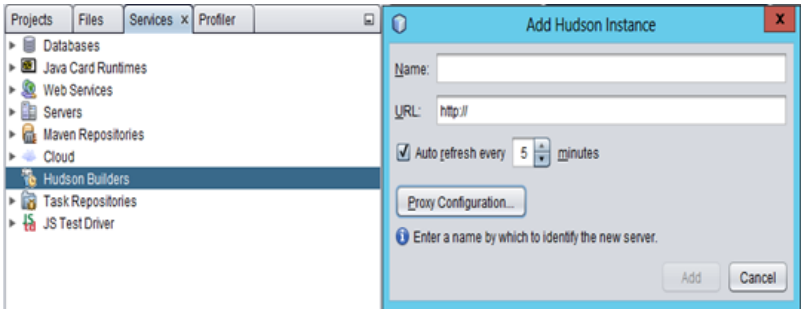

Figura 4: Agregar instancia de servidor Hudson a Netbeans IDE.

Muchos sistemas de control de versiones también pueden ser integrados a Netbeans, entre ellos Git, Mercurial, Subversion, todos a través de plug-ins que se le adicionan al IDE y luego se configuran en las opciones que contiene Netbeans.
En la Figura 5 se muestran algunos de los SCV que pueden ser utilizados desde Netbeans y específicamente Git.

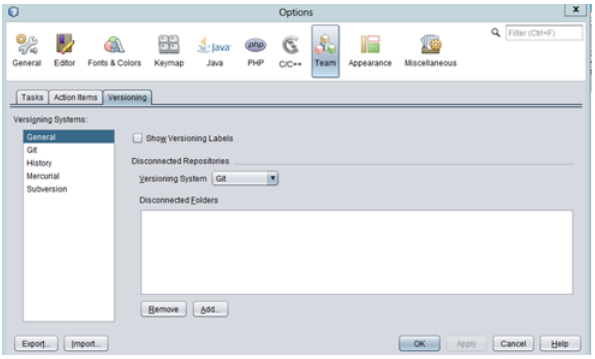

Figura 5: Configurar Sistema de Control de versiones en Netbeans IDE.

INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES
Vol. 3, No. 1, (Enero 2019), 32-44 ISSN 2550-6730

Para medir la calidad de código en el proyecto se proponen utilizar dos herramientas fundamentales que, aunque en el servidor funcionan independientemente como reportes también se pueden integrar entre ellas. Para medir cuánto código cubre una prueba unitaria se utiliza Jacoco, un plug-in que se adiciona al servidor Jenkins que a través de tareas Ant, automatizan el proceso de cobertura de código y muestran los reportes una vez terminado el trabajo de construcción. En la Figura 6 se muestra un ejemplo del código de las tareas Ant para realizar la cobertura de código a pruebas unitarias junto a JUnit.

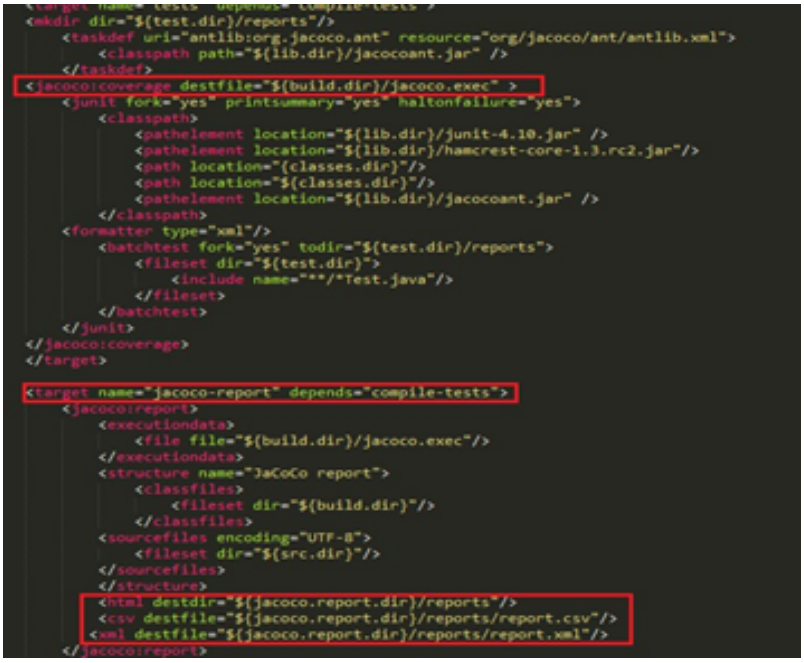

Figura 6: Fichero .xml que contiene las tareas Ant para realizar cobertura de código.

La otra herramienta que se utiliza es SonarQube (Antes Sonar), encargada de la gestión de calidad del código. Su principal objetivo es visualizar las medidas de calidad de los proyectos, recogidas durante el análisis estático del código y la ejecución de pruebas unitarias, de integración y de sistema automatizadas [37]. Sonar se integra fácilmente con el servidor de IC Jenkins a través de un plug-in, además, es necesario tener en ejecución el Sonar-runner para analizar los proyectos sin Maven utilizando SonarQube. SonarQube es una plataforma de código abierto que permite cubrir los aspectos relacionados con la calidad de código. La Figura 7 muestra la configuración de SonarQube en el trabajo de construcción.

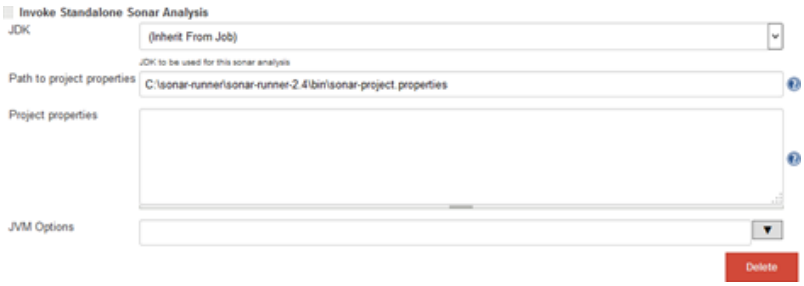

Figura 7: Utilización de Sonar en el trabajo de construcción.

Con las herramientas antes mencionadas se realiza una revisión estática del código, pero respecto a las pruebas unitarias el diseño de los casos de pruebas sigue quedando en manos de los desarrolladores. Para resolver esta situación se desarrolló un plug-in para la generación de pruebas unitarias en lenguaje Java que permite automatizar el diseño de los casos de prueba, contribuyendo a disminuir considerablemente el tiempo y esfuerzo dedicado a la realización de pruebas unitarias en los equipos de desarrollo.
En la Figuras 8 y 9 se puede observar el grafo de control de flujo generado por la herramienta del método seleccionado. Para ello, solo es necesario seleccionar el método y dar clic en el botón de la aplicación "Generar GCF".

Figura 8: Vértices del GCF generado.

Figura 9: Arcos del GCF generado.

Es necesario especificar la descripción del dominio de los parámetros de entrada y la variable de control de cada ciclo, en caso de existir. Una vez hechas las especificaciones necesarias y generado el grafo de control de flujo, se pueden generar los casos de pruebas, con solo seleccionar el submenú "Obtener casos de prueba" que se encuentra en la pantalla principal del plug-in. En la Figura 10 se pueden observar los casos de prueba generados, para los cuales es necesario especificar el resultado esperado como se muestra en la figura.

INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES
Vol. 3, No. 1, (Enero 2019), 32-44 ISSN 2550-6730

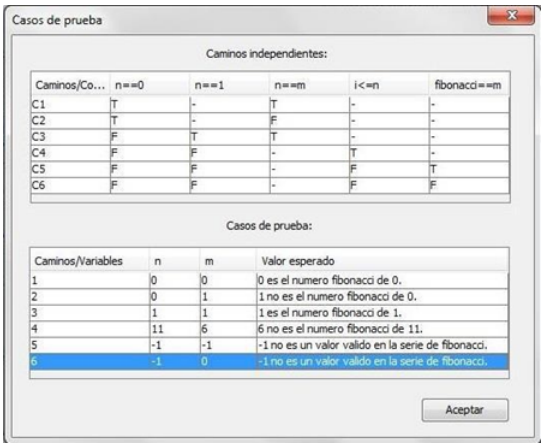

Figura 10: Casos de prueba generados.

Luego, se puede generar el código de pruebas con solo seleccionar el submenú "Crear clase de prueba de JUnit" que se encuentra en la pantalla principal del plug-in. En la Figura 11 se puede observar la clase de prueba, la cual se crea en el proyecto bajo prueba.

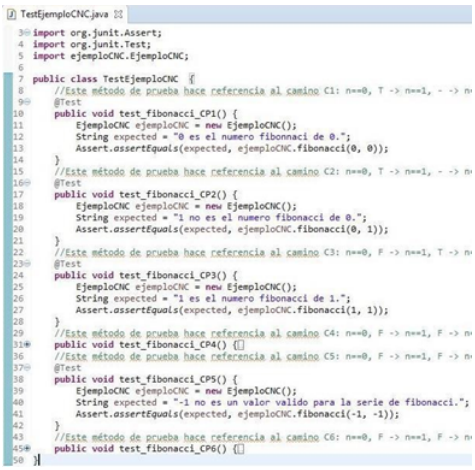

Figura 11: Código de prueba generado.

Finalmente, se puede utilizar la herramienta JUnit que proporciona el propio Eclipse para ejecutar el código de pruebas generado (ver Figura 12).

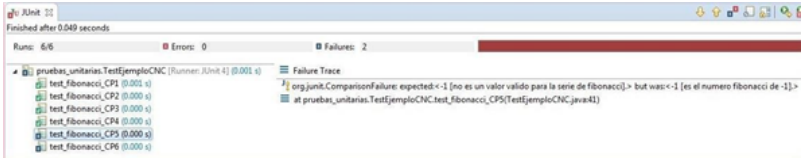

Figura 12: Ejecución del código de prueba generado con la herramienta JUnit.

Para las pruebas funcionales se diseñó un caso de estudio basado en una aplicación real, de la cual se extrajeron 6 funcionalidades que como promedio tienen 3 atributos cada una. Las variables incluidas cubren los tipos de datos: cadena, numérico, enumerado, lógico y fecha.
La Figura 13 muestra el gráfico con el resultado de las ejecuciones del modelo para las seis funcionalidades en correspondencia con el 120 %, 100 % y 80 % de cobertura de escenarios. A partir del análisis de los resultados se pudo comprobar que:

- Se garantiza cubrir los porcentajes de cobertura de los escenarios indicados en cada caso.
- Se generan cantidad de combinaciones de valores mínimas, en los casos de porcentajes de cobertura de hasta 100 %, en particular se genera una única combinación para cada escenario.
- Para los porcentajes de cobertura superiores a 100 % se generan combinaciones que satisfacen un mismo escenario, pero ningún escenario queda sin al menos una combinación de valores.

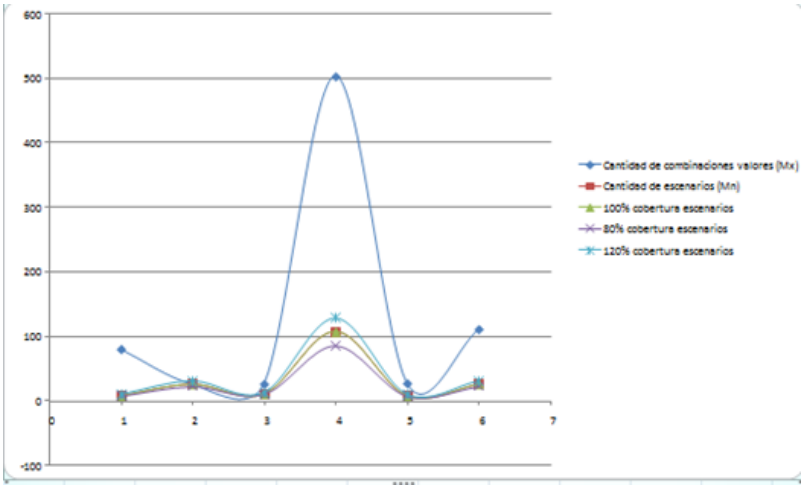

Figura 13: Gráfico de dispersión de generación de combinaciones de valores para diferentes funcionalidades.

4. Conclusiones

El presente trabajo facilita el proceso de pruebas durante el desarrollo de productos de software, debido a que integra la generación y ejecución automática de pruebas en un entorno de desarrollo de aplicaciones mediante la utilización de la práctica de integración continua. Además, permite reducir el tiempo y esfuerzo dedicado por los probadores y diseñadores de casos de prueba en el diseño y ejecución de pruebas. Se recomiendan un grupo de buenas prácticas encaminadas a introducir las pruebas en los entornos productivos como elemento de control de calidad de las aplicaciones desarrolladas en la empresa.

Referencias

[1] GJ Myers, C Sandler y T Badgett. *The art of software testing*. 3rd. New Jersey, 2011. ISBN: 978-1-118-03196-4.

INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES

Vol. 3, No. 1, (Enero 2019), 32-44 ISSN 2550-6730

[2] Ting Chen y col. "State of the art: Dynamic symbolic execution for automated test generation". En: *Future Generation Computer Systems* 29.7 (sep. de 2013), págs. 1758-1773. ISSN: 0167739X. DOI: 10.1016/j.future.2012.02.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167739X12000398.

[3] Frank Elberzhager y col. "Reducing test effort: A systematic mapping study on existing approaches". En: *Information and Software Technology* 54.10 (oct. de 2012), págs. 1092-1106. ISSN: 09505849. DOI: 10.1016/j.infsof.2012.04.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0950584912000894.

[4] A.M. Memon, M.E. Pollack y M.L. Soffa. "Hierarchical GUI test case generation using automated planning". En: *IEEE Transactions on Software Engineering* 27.2 (2001), págs. 144-155. ISSN: 00985589. DOI: 10.1109/32.908959. URL: http://ieeexplore.ieee.org/document/908959/.

[5] L. Morales. "Componente de generación automática de valores de pruebas". Tesis doct. Instituto Superior Politécnico José Antonio Echeverría, 2012.

[6] A. Moreira. "Entorno de integración continua basado en software libre para el desarrollo de aplicaciones Java". Tesis doct. Instituto Superior Politécnico "José Antonio Echeverría", 2016.

[7] Beatriz Lamancha Pérez y Macario Polo. "Generación automática de casos de prueba para Líneas de Producto de Software Automatic test case generation for software product lines". En: *Revista Española de Innovación, Calidad e Ingeniería del Software* 5.2 (2009). ISSN: 1885-4486. URL: http://www.redalyc.org/pdf/922/92217153004.pdf.

[8] Tang Rongfa. "Adaptive Software Test Management System Based on Software Agents". En: 2012, págs. 1-9. DOI: 10.1007/978-3-642-25437-6_1. URL: http://link.springer.com/10.1007/978-3-642-25437-6%7B%5C_%7D1.

[9] Soma Sekhara Babu Lam y col. "Automated Generation of Independent Paths and Test Suite Optimization Using Artificial Bee Colony". En: *Procedia Engineering* 30 (2012), págs. 191-200. ISSN: 18777058. DOI: 10.1016/j.proeng.2012.01.851. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877705812008612.

[10] Ying XING y col. "Intelligent test case generation based on branch and bound". En: *The Journal of China Universities of Posts and Telecommunications* 21.2 (abr. de 2014), págs. 91-103. ISSN: 10058885. DOI: 10.1016/S1005-8885(14)60291-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S1005888514602910.

[11] Zhiqiang Zhang y col. "Generating combinatorial test suite using combinatorial optimization". En: *Journal of Systems and Software* 98 (dic. de 2014), págs. 191-207. ISSN: 01641212. DOI: 10.1016/j.jss.2014.09.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121214001939.

[12] S Bergmann. *Sitio oficial de PHPUnit*. 2015. URL: http://phpunit.de/ (visitado 15-10-2015).

[13] JUnit. *JUnit 5*. URL: https://junit.org/junit5/ (visitado 09-10-2015).

[14] NUnit. *NUnit.org*. URL: http://nunit.org/ (visitado 09-10-2015).

[15] Bestoun S. Ahmed y Kamal Z. Zamli. "Comparison of metahuristic test generation strategies based on interaction elements coverage criterion". En: *2011 IEEE Symposium on Industrial Electronics and Applications*. IEEE, sep. de 2011, págs. 550-554. ISBN: 978-1-4577-1417-7. DOI: 10.1109/ISIEA.2011.6108773. URL: http://ieeexplore.ieee.org/document/6108773/.

[16] Saswat Anand y col. "An orchestrated survey of methodologies for automated software test case generation". En: *Journal of Systems and Software* 86.8 (ago. de 2013), págs. 1978-2001. ISSN: 01641212. DOI: 10.1016/j.jss.2013.02.061. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121213000563.

[17] Fabrice Bouquet y col. "A test generation solution to automate software testing". En: *Proceedings of the 3rd international workshop on Automation of software test - AST '08*. New York, New York, USA: ACM Press, 2008, pág. 45. ISBN: 9781605580302. DOI: 10.1145/1370042.1370052. URL: http://portal.acm.org/citation.cfm?doid=1370042.1370052.

[18] José Carlos Bregieiro Ribeiro. "Search-based test case generation for object-oriented java software using strongly-typed genetic programming". En: *Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation - GECCO '08*. New York, New York, USA: ACM Press, 2008, pág. 1819. ISBN: 9781605581316. DOI: 10.1145/1388969.1388979. URL: http://portal.acm.org/citation.cfm?doid=1388969.1388979.

[19] Gustavo Carvalho y col. "NAT2TESTSCR: Test case generation from natural language requirements based on SCR specifications". En: *Science of Computer Programming* 95 (dic. de 2014), págs. 275-297. ISSN: 01676423. DOI: 10.1016/j.scico.2014.06.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167642314002858.

[20] Eugenia Díaz y col. "A tabu search algorithm for structural software testing". En: *Computers & Operations Research* 35.10 (oct. de 2008), págs. 3052-3072. ISSN: 03050548. DOI: 10.1016/j.cor.2007.01.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0305054807000214.

[21] Roger Ferguson y Bogdan Korel. "The chaining approach for software test data generation". En: *ACM Transactions on Software Engineering and Methodology* 5.1 (ene. de 1996), págs. 63-86. ISSN: 1049331X. DOI: 10.1145/226155.226158. URL: http://portal.acm.org/citation.cfm?doid=226155.226158.

[22] Mark Harman. "Automated Test Data Generation using Search Based Software Engineering". En: *Second International Workshop on Automation of Software Test (AST '07)*. IEEE, mayo de 2007, págs. 2-2. ISBN: 978-1-5090-8886-7. DOI: 10.1109/AST.2007.4. URL: https://ieeexplore.ieee.org/document/4296713/.

[23] Mark Harman, S. Afshin Mansouri y Yuanyuan Zhang. "Search-based software engineering: Trends, techniques and applications". En: *ACM Computing Surveys* 45.1 (nov. de 2012), págs. 1-61. ISSN: 03600300. DOI: 10.1145/2379776.2379787. URL: http://dl.acm.org/citation.cfm?doid=2379776.2379787.

[24] I. Hermadi, C. Lokan y R. Sarker. "Dynamic stopping criteria for search-based test data generation for path testing". En: *Information and Software Technology* 56.4 (abr. de 2014), págs. 395-407. ISSN: 09505849. DOI: 10.1016/j.infsof.2014.01.001. URL: https://linkinghub.elsevier.com/retrieve/pii/S0950584914000123.

[25] Muhammad Zohaib Iqbal, Andrea Arcuri y Lionel Briand. "Empirical investigation of search algorithms for environment model-based testing of real-time embedded software". En: *Proceedings of the 2012 International Symposium on Software Testing and Analysis - ISSTA 2012*. New York, New York, USA: ACM Press, 2012, pág. 199. ISBN: 9781450314541. DOI: 10.1145/2338965.2336777. URL: http://dl.acm.org/citation.cfm?doid=2338965.2336777.

[26] Laura Lanzarini, Juan Pablo y La Battaglia. "Dynamic Generation of Test Cases with Metaheuristics". En: *Optimization* 10.2 (2010), págs. 91-96.

[27] C.C. Michael, G. McGraw y M.A. Schatz. "Generating software test data by evolution". En: *IEEE Transactions on Software Engineering* 27.12 (2001), págs. 1085-1110. ISSN: 00985589. DOI: 10.1109/32.988709. URL: http://ieeexplore.ieee.org/document/988709/.

[28] Ankur Pachauri y Gursaran Srivastava. "Automated test data generation for branch testing using genetic algorithm: An improved approach using branch ordering, memory and elitism". En: *Journal of Systems and Software* 86.5 (mayo de 2013), págs. 1191-1208. ISSN: 01641212. DOI: 10.1016/j.jss.2012.11.045. URL: https://linkinghub.elsevier.com/retrieve/pii/S0164121212003263.

INFORMÁTICA Y SISTEMAS
REVISTA DE TECNOLOGÍAS DE LA INFORMÁTICA Y LAS TELECOMUNICACIONES

Vol. 3, No. 1, (Enero 2019), 32-44 ISSN 2550-6730

[29] Roy P. Pargas, Mary Jean Harrold y Robert R. Peck. "Test-data generation using genetic algorithms". En: *Software Testing, Verification and Reliability* 9.4 (dic. de 1999), págs. 263-282. ISSN: 0960-0833. DOI: 10.1002/(SICI)1099-1689(199912)9:4<263::AID-STVR190>3.0.CO;2-Y. URL: http://doi.wiley.com/10.1002/%7B%5C%%7D28SICI%7B%5C%%7D291099-1689%7B%5C%%7D28199912%7B%5C%%7D299%7B%5C%%7D3A4%7B%5C%%7D3C263%7B%5C%%7D3A%7B%5C%%7D3AAID-STVR190%7B%5C%%7D3E3.0.CO%7B%5C%%7D3B2-Y.

[30] R. S. Pressman. *Software Engineering: A Practitioner's Approach*. 7th. New York, USA: McGraw-Hill, 2010. ISBN: 978–0–07–337597–7.

[31] Praveen Ranjan Srivatsava, B. Mallikarjun y Xin-She Yang. "Optimal test sequence generation using firefly algorithm". En: *Swarm and Evolutionary Computation* 8 (feb. de 2013), págs. 44-53. ISSN: 22106502. DOI: 10.1016/j.swevo.2012.08.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S2210650212000612.

[32] Abdelilah Sakti, Yann-Gaël Guéhéneuc y Gilles Pesant. "Boosting Search Based Testing by Using Constraint Based Testing". En: 2012, págs. 213-227. DOI: 10.1007/978-3-642-33119-0_16. URL: http://link.springer.com/10.1007/978-3-642-33119-0%7B%5C_%7D16.

[33] Sapna Varshney y Monica Mehrotra. "Search based software test data generation for structural testing". En: *ACM SIGSOFT Software Engineering Notes* 38.4 (jul. de 2013), pág. 1. ISSN: 01635948. DOI: 10.1145/2492248.2492277. URL: http://dl.acm.org/citation.cfm?doid=2492248.2492277.

[34] Joachim Wegener, Andre Baresel y Harmen Sthamer. "Evolutionary test environment for automatic structural testing". En: *Information and Software Technology* 43.14 (dic. de 2001), págs. 841-854. ISSN: 09505849. DOI: 10.1016/S0950-5849(01)00190-2. URL: http://linkinghub.elsevier.com/retrieve/pii/S0950584901001902.

[35] Vogella. *Eclipse, Android and Java training and support*. URL: https://www.vogella.com/ (visitado 09-10-2015).

[36] Paul M. Duvall, Steve Matyas y Andrew Glover. *Continuous Integration: Improving Software Quality and reducing risks*. Ed. por Inc. Pearson Education. Second Pri. Boston, 2007. ISBN: 13: 978-0-321-33638-5.

[37] Sonarqube. *Continuous Inspection | SonarQube*. URL: https://www.sonarqube.org/ (visitado 12-12-2018).

5. Sobre los autores

Danay Larrosa Uribazo, Ingeniera Informática de la Facultad Ingeniería Informática, Universidad Tecnológica de la Habana José Antonio Echeverría (CUJAE). Investiga en los temas de calidad de software, y la generación de pruebas unitarias automáticas.

Sandra Verona Marcos, profesora instructora, Facultad de Ingeniería Informática, Universidad Tecnológica de la Habana José Antonio Echeverría (CUJAE), Máster en Informática Aplicada. Se ha desempeñado como docente, investigando en los temas de metodologías de desarrollo ágil, calidad de software, y la generación de pruebas automáticas.

Perla Fernández Oliva, profesora asistente, Facultad de Ingeniería Informática, Universidad Tecnológica de la Habana José Antonio Echeverría (CUJAE), Máster en Informática Aplicada. Se ha desempeñado como docente, investigando en los temas de inteligencia artificial, calidad de software, y la generación de pruebas automáticas.

Martha Dunia Delgado Dapena, Universidad Tecnológica de la Habana José Antonio Echeverría (CUJAE), Doctora en Ciencias Técnicas. Profesora Titular. Se ha desempeñado como docente por más de 15 años, investigando en los temas de calidad de software, y la generación de pruebas automáticas.