Pruebas funcionales automatizadas para evaluar la calidad en productos software, implementando métodos de Testing no heurístico

Cesar Yesid Barahona Rodríguez (*Author*)
Docente Universidad de Cundinamarca
Bogotá, Colombia
cbarahona@ucundinamarca.edu.co

Stevenson Marquez Rangel (*Author*)
Estudiante Universidad de Cundinmacarca
Bogotá, Colombia
stevenson@openmailbox.org

Johan Suarez Campos (*Author*)
Estudiante Universidad de Cundinmacarca
Bogotá, Colombia
stevenson@openmailbox.org

***Abstract*— Este articulo expone como se implementaron métodos automatizados de Testing no heurísticos en la plataforma CALISOFT (Calidad de Software) de la Universidad de Cundinamarca. Esta plataforma contiene diferentes módulos que evalúan la estructura de un producto software para así certificar su calidad frente al mercado, actualmente la plataforma cuenta con un módulo que realiza pruebas funcionales de forma automática gracias a la implementación de dos componentes que se explicarán en este artículo, este proceso de automatización se implementa con el objetivo de realizar evaluaciones más precisas, rápidas y con un alto grado de eficacia.**

Keywords*— *Pruebas funcionales, Testing, métodos no heurísticos, validación, automatización.

I. INTRODUCCIÓN

Este artículo se basa en explicar la implementación y el funcionamiento de pruebas funcionales automatizadas implementadas en procesos de evaluación de calidad sobre productos software, con el objetivo de aumentar la calidad y disminuir tiempos de ejecución. Se implementaron una serie de pruebas funcionales automatizadas sobre un módulo de Testing proveniente del software CALISOFT (Calidad de software) desarrollador en la Universidad de Cundinamarca, el proceso consiste en implementar métodos de Testing no heurístico, que es un método de evaluación de calidad proveniente de la ingeniería de la usabilidad [1].

II. ANTECEDENTES.

A la fecha se considera cualquier actividad tecnológica como un factor crítico en la vida del ser humano, dado que la tecnología ha avanzado tan rápidamente que muchas actividades cotidianas han pasado a ser suplementadas por software que reducen tiempo y costos, por ejemplo, en transacciones electrónicas, telemedicina, negocios empresariales y entre otras más. Un error en el software puede generar grandes complicaciones en la vida del ser humano [2].

Los siguientes son ejemplos de mal funcionamiento de un software en condiciones no favorables:

- El 4 de junio de 1996, ocurrió un error en el primer vuelo del cohete Ariane 5, el cohete perdió amplitud y orientación causado por el fallo del motor pocos segundos después del lanzamiento, generando una explosión y pérdida de información. Esto se debió al hecho de que el código con el que se programó el despegue, fue reutilizado de su antecesor, el Ariane 4, el código no era compatible con el nuevo modelo y al no realizar pruebas de software, el error ocasionó la destrucción de la base del lanzamiento. [3]

- Entre los años 1985 a 1987 en Canadá, ocurrieron una de accidentes a causa de la máquina Therac-25, que empleaba radiación para realizar terapias, los incidentes dejaron a varias personas muertas y a otros severamente lesionado. La raíz del problema, un error en la programación de la interfaz gráfica, el software permitía trabajar con altas radiaciones que el cuerpo humano no puede soportar.[4] El software antes de producción debía ser evaluado con múltiples entradas de datos, lo cual no se realizó cayó en demandas.

III. IMPORTANCIA DE LAS PRUEBAS FUNCIONALES

La seguridad es el factor principal, dentro del modelo de requerimientos se encuentra como un atributo no funcional directamente responsable de la calidad del

software, conservando y previniendo la pérdida de información conservando la integridad de los datos [5].

Basados en la norma ISO 9000, la definición de calidad es entendida como un conjunto de propiedades o características inherentes de un producto, que cumplen con una serie de requisitos planteados y que logran satisfacer las necesidades de un usuario [5].

Para mitigar los riesgos y disminuir perdidas tanto económicas y de clientes, la industria del software le da importancia al proceso de pruebas funcionales que certifiquen la calidad del producto final, fortaleciendo la entidad y aumentando su competitividad [6].

IV. PRUEBAS DE SOFTWARE

Las pruebas son las encargadas de verificar y controlar el buen funcionamiento del software, para ello se debe llevar un orden correcto para alcanzar los objetivos propuestos y así cumplir con los requerimientos del sistema, se debe empezar por la creación de casos de prueba, cuya finalidad es observar y prever como será el funcionamiento y las respuestas adecuadas que debe dar el sistema frente a múltiples acciones. Los resultados determinan si el sistema cuenta con errores en su programación, por ende, en el código del software. Hay estudios donde los casos de prueba son obtenidos mediante los modelos de casos de uso que se crean al inicio de todo proyecto, además de usar otro componente en la arquitectura que son los requerimientos del sistema, y también aconsejan realizar un cruce entre ambos para obtener una mejor definición de los casos de prueba. [7].

Cabe resaltar que, en este proyecto de automatización de pruebas funcionales, los casos de prueba que se realizan en el sistema son obtenidos mediante el estudio de los modelos de casos de uso más los requerimientos del proyecto a evaluar.

El principal desafío es evaluar un software en el menor tiempo posible, con el objetivo de mantener la calidad del producto software siempre en un buen estándar y entregarlo en un ciclo corto, como solución a esta problemática se utilizan pruebas automatizadas que reduzcan los tiempos de entrega y que optimice la calidad del producto a evaluar. [8]

V. PLANTAMIENTO DEL PROBLEMA

Actualmente en la Universidad de Cundinamarca existe un software llamado CALISOFT, este software tiene como propósito evaluar la calidad de un producto software con base a su documentación, código base del software, nomenclatura de la base de datos y un módulo que evalúa los parámetros de entrada que puedan recibir los múltiples formularios del software a evaluar. Anteriormente durante el proceso de evaluación, el evaluador tardaba entre dos o tres días para realizar correctamente las pruebas de calidad sobre los formularios, cada parámetro de entrada usado era dado por la creatividad o experiencia del evaluador, esto ocasionaba que no todos los productos software se evaluaran de la misma manera, dado que cada evaluador tenía su propia manera de evaluar, además, habían casos en los que no se probaban de todas las formas posibles o necesarias los diferentes campos de entrada de los formularios.

Como solución a esta problemática se hace una implementación de métodos de Testing no heurísticos programados en el módulo de Testing, con la finalidad de reducir tiempos de ejecución y que el evaluador ya no tenga que disponer de sus conocimientos para realizar pruebas dado que los parámetros son programados y la evaluación se hace de forma automática.

VI. IMPLEMENTACIÓN

Para su implementación se crearon dos tablas en la base de datos, que son las siguientes;

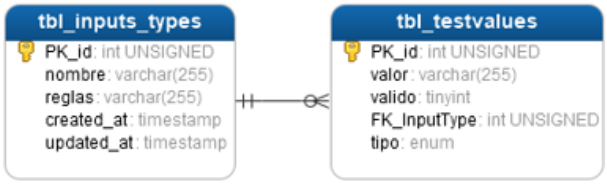

Fig. 1 Tablas destinadas a almacenar la información que se usará en el proceso de Testing automatizado.

En la tabla “Tbl_inputs_types” se almacenan los diferentes tipos de inputs que puede haber en un formulario, por ejemplo, un tipo email, contraseña, text, number, etc. Entre sus campos se puede guardar el nombre del tipo, las reglas con las que se valida la información y los datos que ingresan mediante ese tipo de inputs y dos campos que almacenan la fecha en que se crea y la fecha en que se realice una modificación desde el sistema. Y la tabla “Tbl_testvalues” almacena los diferentes valores con los que trabajará el proceso de Testing no heurístico, en el campo de valor se asignará una sentencia que corresponda al tipo de input en el que se utilizará, por ejemplo, para el tipo email, en valor se almacenará una sentencia como; prueba@email.com o

pruebaemail.com, como se puede observar, una sentencia es correcta y la otra es incorrecta, ese valor de incorrecto e incorrecto se almacena en el cambo de la tabla “valido”, donde 1 será correcto y 0 incorrecto. La columna FK_InputType corresponde a la llave foránea de la tabla Tbl:inputs:types, y el campo llamado “tipo” corresponde a al tipo de ataque, si es html, xss o sql.

El proceso cuenta con la intervención de dos actores, un evaluador y un desarrollador. Como paso inicial, el desarrollador debe subir toda la documentación de su proyecto, dentro de la documentación se encontrarán los diagramas de casos de uso, con base a los diagramas de casos de uso, el evaluador procede a crear casos de prueba donde vea que hay un ingreso de datos al software del desarrollador.

Fig. 2 Formulario de creación de un caso prueba.

Con los casos de prueba creados, el desarrollador tiene como objetivo dirigirse al formulario que corresponda y crear una copia del formulario por medio de la extensión de Calisoft.

Fig. 3 Interfaz d de Calisoft que ayuda en la obtención de atributos de los inputs encontrados en un formulario en código Html.

La extensión se sitúa en el apartado del inspector de páginas, el cual se accede oprimiendo clic en la ventana del navegador y seleccionando la opción “Inspeccionar”. El desarrollador debe buscar el formulario correspondiente al caso prueba y debe dar clic en el botón “Seleccionar” de la extensión para que se dibuje un recuadro sobre el formulario, al dar clic en recuadro la extensión guardará todos los atributos de los inputs que encuentre en el formulario y de esa forma se crea una copia del formulario.

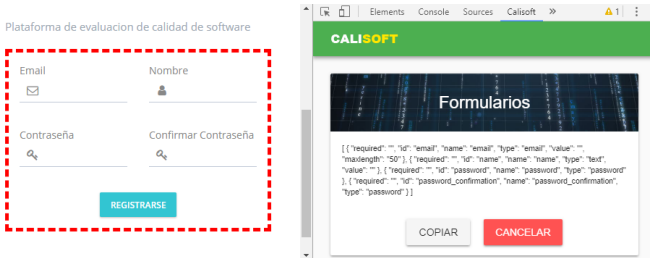

Fig. 4 Uso de la extensión de Calisoft para extraer los atributos de los inputs de un formulario destinado al registro de un usuario.

Cabe resaltar que la copia se almacena como un JSON.

Cuando el caso de prueba es actualizado con el formulario del desarrollador, el evaluador procede a realizar el testeo no heurístico.

Fig. 5 Módulo de Testing. En este módulo se recrea la copia del formulario que un desarrollador sube a la plataforma, en el cual se procede a ejecutar las pruebas no heurísticas automatizadas para el proceso de evaluación de calidad.

El sistema realiza una interpretación de los inputs del formulario y únicamente del evaluador debe presionar un botón para que el sistema ingrese datos que obtiene desde la base de datos, esto se logra gracias a la herramienta proporcionada por el framework Laravel llamada; Laravel Faker, "Es un asistente que ingresa datos existentes en la base de datos que coinciden con un conjunto determinado de criterios"[9]. Cada sentencia que viene de la base de datos tiene un atributo que se llama "valido", de tipo booleano donde un 0 representa que la sentencia corresponde a un dato incoherente o malicioso, y un 1 representa que la sentencia está bien digitada para el input sobre el cual se va a probar y además no es una sentencia maliciosa para el sistema. Resalto que uso el término "Malicioso" para hacer referencia a una sentencia destinada a ataques de Sql o Xss que pueden averiar el funcionamiento del software si se le permite ingreso a la base de datos.
Por el lado del input que el sistema recrea, este viene acompañado de una serie de reglas, que son las mismas que el desarrollador cuenta en su aplicativo web.

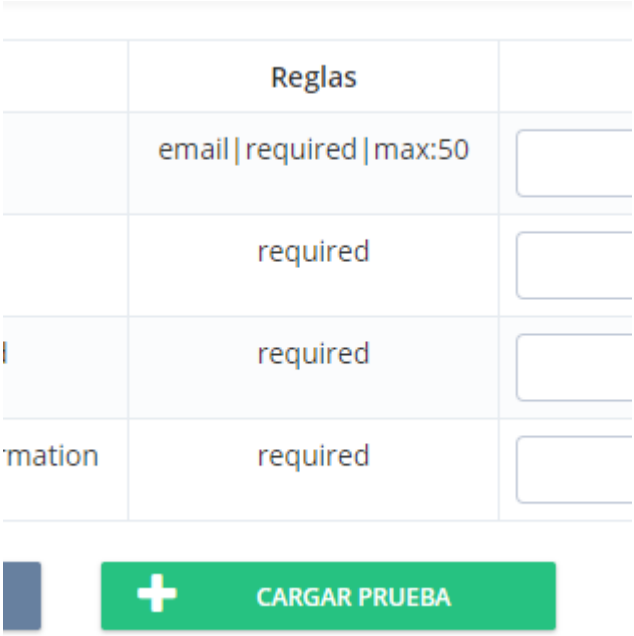

Fig. 6 Reglas provenientes del formulario. Estas reglas se usan para corroborar la correcta validación que implementó el desarrollador en su formulario.

Estas reglas se combinan con un plugin de Vue.js llamado Vee-Validate, cuyo objetivo es validar campos de entrada y mostrar errores de una forma fácil y potente"[10]. Lo cual hace que recree esas mismas reglas sobre el input y que además si se dispara una de ellas, aparecerá un mensaje en de color rojo debajo el input.

Fig. 7 Plugin Vee-validate. El plugin se activa al momento en que permite el ingreso de información errónea o maliciosa.

El sistema cuenta con una programación booleana que al momento de ingresar un dato y si el sistema no muestra ningún error por medio del componente de javascript, almacenará un 0, en cambio, si el componente se dispara y muestra un error, el sistema almacenará un 1.

El método de calificación es el siguiente; se emplea una lógica simbólica denominada como "Disyunción exclusiva y operador OR exclusivo (XOR)", se denota

por el símbolo $\oplus$, donde un valor es verdadero cuando dos preposiciones q y p son contrarias y si q y p son dos preposiciones iguales, el valor obtenido es falso [11].

TABLA I
DISYUNCIÓN EXCLUSIVA Y OPERADOR OR EXCLUSIVO (XOR)

Sentencia traída de base de datos (p)	Error proveniente del componente de Javascript (q)	Calificación (p $\oplus$ q)
0	1	1
1	0	1
0	0	0
1	1	0

Si la sentencia proveniente de la base de datos (p) es correcta, se denota por un 1, este valor está almacenado en el campo "valido" que anteriormente se mencionó, si no se dispara ningún error al ingresar ese dato, el sistema registrará un 0 para ese error, que significa que no encontró un error en la sentencia para ese tipo de input, implementando XOR obtenemos como resultado un 1, que equivale a una calificación correcta. Esta calificación se puede apreciar en la columna "Estado" en la figura 7.

Por cada inserción de la base de datos sobre un input se aplica la lógica XOR para determinar sí está bien o mal las validaciones que uso el desarrollador. Cada prueba tiene una calificación sobre el formulario, la calificación va ligada a la cantidad de inputs que el formulario posea, se toma un valor de 100% y se divide en la cantidad de inputs, eso le dará a cada input un porcentaje, por ejemplo, si un formulario tiene cuatro inputs, cada input tendrá un valor de 25%, si la calificación obtenida por la lógica de XOR es positiva, el valor se cuenta como un 25 entero, pero si es negativa, la calificación (nota) se toma como 0, al final se suman todos los resultados como se aprecia en la formula (1).

(1)

$$Nota = \sum_{i=0}^{CantidadInputs} \frac{100}{CantidadInputs} \times ResultadoXOR$$

Una vez obtenida la calificación parcial de cada prueba, se procede a realizar una sumatoria de cada calificación parcial y dividirla sobre el número total de calificaciones parciales que se hicieron un formulario en específico, de ese modo, se obtiene la calificación final para dicho formulario, este proceso corresponde a la formula (2).

(2)

$$NotaFinal = \frac{\sum CalificacionParcial}{CantidadDeCalificacionesParciales}$$

El valor obtenido es guardado en la tabla donde está creado el caso prueba del correspondiente formulario en el apartado de "calificación".

VII. CONCLUSIONES

- La automatización de las pruebas permite que haya un estándar en el proceso de evaluación, ya no es necesario la creatividad o experiencia del evaluador para probar múltiples opciones datos correctos e incorrectos sobre cualquier formulario.

- La calidad de evaluación aumentó, dado que se disminuyó tiempos de evaluación, tiempos de respuesta y se optimizaron procesos que anteriormente eran un poco largos y complejos.

- Para la elaboración de este módulo, se implementó dos plugin que son base en su funcionamiento, el plugin Vee.Validate de Vue.js, encargado de validar en tiempo real un input con base a las reglas de validación implementadas por el desarrollador, y el plugin Laravel Faker, encargado de ingresar información proveniente de la base de datos de manera automática, este proceso se realiza sobre los sobre los inputs obtenidos del formulario que ofrece el desarrollador.

VIII. REFERENCIAS

[1] M. Bolaños-Pizarro and R. Vidal-Infer, A; Navarro-Molina, Carolina; Valderrrama-Zuriañ, Juan Carlos; Aleixandre-Benavent, "Usabilidad: concepto y aplicaciones en las páginas web médicas," *Papeles Médicos*, vol. Vol 16, Nr, no. 1, pp. 14–21, 2007.

[2] M. E. López Inga and R. M. Guerrero Huaranga, *Modelo de inteligencia de negocios y analítica en la nube para pymes del sector retail en Perú*, vol. 12, no. 20. Universidad Cooperativa de Colombia, 2018.

[3] LIONS J. L., "ARIANE 5 Failure - Full Report," 1996. [Online]. Available: http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html. [Accessed: 19-Apr-2018].

[4] N. G. Leveson and C. S. Turner, "An Investigation of the Therac-25 Accidents," *Computer (Long. Beach. Calif).*, vol. 26, no. 7, pp. 18–41, Jul. 1993.

[5] Y. Fundación Universitaria Luis Amigó. Facultad de Ingenierías. and V. Vega, *PRUEBAS DE SEGURIDAD: ESTUDIO DE HERRAMIENTAS*, no. 17. Fundación Universitaria Luis Amigó, Facultad de Ingenierías, 2009.

[6] M. L. Rojas-Montes, F. J. Pino-Correa, and J. M. Martínez, *Proceso de pruebas para pequeñas organizaciones desarrolladoras de software*, vol. 24, no. 39. Universidad Pedagógica y Tecnológica de Colombia, 2015.

[7] G. Kaplan *et al.*, "Generación semi automática de casos de prueba a partir de escenarios," May 2015.

[8] M. A. Mascheroni, M. K. Cogliolo, and E. Irrazábal, "Automatización de pruebas de compatibilidad web en un entorno de desarrollo continuo de software," Nov. 2016.

[9] Laravel, "Database Testing - Laravel - The PHP Framework For Web Artisans," 2016. [Online]. Available: http://laravel.com/docs/master/database. [Accessed: 26-Apr-2018].

[10] Vue.js, "Vee Validar." [Online]. Available: https://vee-validate.logaretm.com/. [Accessed: 26-Apr-2018].

[11] C. Marcial, R. Carmona, and R. T. Milenio, "Matemáticas I."