Automatización de Pruebas para Interfaces de Aplicaciones Web

Miguel Giménez
Ingeniería Informática
Facultad Politécnica
Universidad Nacional de Asunción
Campus Universitario, San Lorenzo
miguel.a.gimenez@gmail.com

Alberto Espínola
Ingeniería Informática
Facultad Politécnica
Universidad Nacional de Asunción
Campus Universitario, San Lorenzo
bettopindu@gmail.com

***Resumen*—El *testing* de *software* es una tarea primordial para asegurar la calidad de un producto de *software*, sin embargo, el *testing* posee un costo elevado en términos de tiempo y recursos. La automatización de pruebas resulta una opción interesante para ejecutar pruebas funcionales para validación de interfaces de aplicaciones *web*, donde se deben producir automáticamente los valores a ser ingresados en los formularios, teniendo en cuenta el contenido dinámico de las páginas web generados por eventos (Ajax). Este trabajo propone una herramienta que busca satisfacer los requerimientos mencionados, proporcionando una estrategia para realizar pruebas de interfaces *web*, implementando dos enfoques conocidos en el área del testing: el *black box testing* y el *white box testing*.**

I. Introducción

Actualmente, muchos de los productos (*software*) utilizados por empresas, están construidos como aplicaciones web, y para realizar *testing* en ellas, mayormente se utilizan herramientas que registran las acciones correctas realizadas por un usuario en el navegador (*browser*) en archivos de órdenes (*scripts*), para luego volver a ejecutarlos y verificar si las modificaciones realizadas produjeron efectos no deseados o incumplan los requerimientos especificados.

Las aplicaciones *web* actuales poseen interfaces enriquecidas (*RIA - Rich Internet Aplications*). Estas interfaces están mayormente desarrolladas con herramientas basadas en JavaScript[1], utilizando Ajax (*Asynchronous JavaScript And XML*)[2] para invocar eventos que a su vez pueden crear nuevos componentes en la página dinámicamente.

Este trabajo propone el desarrollo de una herramienta capaz de ejecutar pruebas de interfaces dinámicas de aplicaciones web (basadas en Javascript y Ajax), adaptándose a los cambios que puedan surgir en dichas interfaces en su ejecución. Se emplearán conceptos definidos en el área de *testing*, como el enfoque *black box testing* y el enfoque *white box testing*, que serán explicados en la siguiente sección. Finalmente se hará una comparación de los resultados obtenidos en ambos enfoques para determinados escenarios de prueba.

El contenido de este documento está compuesto por las siguientes secciones:

II **Estado del arte:** Panorama actual del área de *testing* y su automatización.
III **Definición del problema:** Descripción del problema.
IV **Objetivos del trabajo:** Objetivos generales y específicos del trabajo.
V **Modelo propuesto:** Descripción de la propuesta de solución.

II. Estado del arte

II-A. Definición

El *Software Testing* proporciona información acerca de la calidad del producto o servicio bajo *testing* [3]. El proceso de análisis permite detectar las diferencias entre las condiciones existentes y las requeridas (es decir, errores) [4], a fin de garantizar calidad, verificar y validar, o estimar la fiabilidad del producto de *software* [5]. La verificación y la validación, son las tareas realizadas por los equipos de *testing*, asegurando así la conformidad del producto final basada en altos estándares de calidad. Se define a continuación:

- **Verificación:** Proceso de evaluación de un sistema o componente para determinar si los productos de una fase de desarrollo dada satisfacen las condiciones impuestas al inicio de dicha fase [6].
- **Validación:** Proceso de evaluar un sistema o componente durante o al final del proceso de desarrollo para determinar si este satisface los requerimientos especificados [6].

II-B. La necesidad de realizar Testing

Los usuarios de los sistemas exigen que el software adquirido o desarrollado realice sus funciones y además cumpla niveles mínimos de calidad. Para ello, durante el proceso de desarrollo del software, el ingeniero de *testing* debe presentar un plan de pruebas que permita validar que el *software* cumpla con el nivel de calidad solicitado.

Encontrar y reparar un problema de *software* después de la entrega es a menudo 100 veces más caro que encontrar y corregir durante la fase de especificación de requerimientos y la fase de diseño del *software* [7].

[1]Es un lenguaje de programación interpretado, dialecto del estándar ECMAScript [1].

[2]Conjunto de tecnologías intregadas para mejorar la interactividad, velocidad y usabilidad de las aplicaciones *web* [2].

II-C. Enfoques de Testing

Dos enfoques más populares de *testing* son las llamadas *Black Box Testing* y *White Box Testing*, que describen los puntos de vista que el ingeniero de *testing* deberá tener en cuenta a la hora de diseñar sus casos de pruebas. Se describe la diferencia básica entre dichos enfoques:

- **Black Box Testing:** También llamada prueba funcional, es la prueba que ignora el mecanismo interno de un sistema o componente y se centra exclusivamente en los resultados generados en respuesta a las entradas seleccionadas y las condiciones de ejecución [6].
- **White Box Testing:** También llamada prueba estructural o pruebas de caja de cristal, es la prueba que tiene en cuenta el mecanismo interno de un sistema o componente [6], es decir, su arquitectura y código fuente.

II-D. Niveles de Testing

Existen varios niveles de *testing* que deberían realizarse en un producto de *software*. Dos consideraciones a tener en cuenta para identificar el nivel de pruebas; es la opacidad de visión del *tester* del *software* (si es *Black Box* o *White Box*) y la otra es la escala de prueba (si el *tester* está examinando un pequeño fragmento del código o todo el sistema y su entorno).

- **Prueba Unitaria** (*White Box* - Código): *Testing* aplicado a unidades individuales o grupos relacionados de *software* o *hardware* [6].
- **Prueba de Integración** (*Black Box/White Box* - Código/Sistema): *Testing* aplicado a componentes de *software*, *hardware*, o ambos, son combinados y probados para evaluar la interacción entre ellos [6].
- **Prueba de Sistema** (*Black Box* - Sistema): *Testing* llevado a cabo en un sistema completo e integrado para evaluar la conformidad del sistema con sus requisitos especificados [6].

II-E. Pruebas Funcionales

Las pruebas funcionales son realizadas según la especificación de requerimientos para planificar los casos de prueba y asegurar que el código hace lo que se pretendía que realice. A continuación se describen los tipos de pruebas funcionales:

- **Pruebas de Aceptación:** Es un *testing* formal para determinar si un sistema satisface o no, el criterio de aceptación (el criterio que satisface que el cliente acepte el sistema) y permitiendo al cliente determinar si va a aceptar el sistema [6].
- **Pruebas de Regresión:** Es la reejecución de *testing* selectiva de un sistema o componente para verificar que las modificaciones realizadas no hayan causado efectos involuntarios y que dicho sistema o componente todavía cumple con sus requerimientos especificados [6].
- **Pruebas Alfa:** Las pruebas alfa pueden ser realizadas por usuarios o clientes potenciales, o por un equipo de prueba independiente en lugar de los desarrolladores. Se emplean a menudo como una forma interna de pruebas de aceptación, antes de que el *software* pase a las pruebas beta.
- **Pruebas Beta:** Cuando una avanzada versión parcial o total de un paquete de *software* ya se encuentra disponible, usuarios potenciales o *testers beta* instalan el *software* y lo utilizan libremente, como lo desean, reportando los errores descubiertos durante el uso.

II-F. Automatización de Pruebas (Automation Testing)

La automatización de pruebas consiste en el uso de un *software* especial (separado del *software* a ser probado) para controlar la ejecución de las pruebas y la comparación de los resultados reales con los resultados esperados [8].

1) Evolución de la automatización de pruebas: Para comprender la automatización de pruebas, es importante conocer la evolución de las técnicas de automatización. A continuación se cita la evolución de las técnicas de *testing* clasificadas por generaciones [9]:

- Primera Generación: Grabar y Reproducir.
- Segunda Generación: Reutilización de funciones en scripts de *testing*.
- Tercera Generación: *Scripts*/Funciones enfocadas a Datos.
- Cuarta Generación: *Scripts*/Funciones basadas en Acciones.
- Quinta Generación: Automatización sin *Scripts*.

Para que una herramienta de automatización pertenezca a la quinta generación, no debe ocurrir lo siguiente:

- El ingeniero de *testing* no debe definir *scripts* de automatización de pruebas.
- El desarrollador de pruebas no debe implementar funciones o casos de prueba (*action-words*).

Actualmente existen las siguientes herramientas comerciales de quinta generación [9]: TestArchitect [3], Unified TestPro (UTP) [4], Certify [5].

2) Enfoque híbrido de automatización de pruebas: Una alternativa a lo anterior es un enfoque híbrido, en el cuál las herramientas poseen características de múltiples generaciones, indicadas a continuación:

- **Prueba guiada por código (*Code-Driven Testing, CDT*):** Generalmente las interfaces públicas de las clases, módulos o bibliotecas se prueban con una

[3]Herramienta que expande las funcionalidades de TestFrame Engine 5, un controlador escrito en C++ e implementado como un DLL de Microsoft Windows, sirve para programar las funciones de *actionwords* [9]. Web: http://testarchitect.logigear.com/

[4]Se basa en TestFrame Engine 5 de Logica CMG, cuya empresa ya no existe, fue adquirida por CGI (http://www.cgi.com) en el 2008. Extendido con nuevas funcionalidades. Web: http://www.sdtcorp.com/

[5]Herramienta diseñada por Linda Hayes, de acuerdo a la acumulación de librerías con el pasar de los años, dando como resultado funciones genéricas reutilizables. Propiedad de Worksoft. Web: http://www.worksoft.com/products/worksoft-certify.html

variedad de argumentos de entrada para confirmar que los resultados devueltos son correctos.

- **Prueba de interfaz gráfica de usuario (*Graphic User Interface Testing, GUI Testing*):** Consiste en la generación de eventos de interfaz de usuario, tales como pulsaciones de teclado y clics del ratón, y observar los cambios que resultan en la interfaz de usuario, para validar que el comportamiento observado del programa sea el correcto.
- **Prueba guiada por datos (*Data-Driven Testing, DDT*):** Consiste en la definición de *scripts*, donde se permiten definir variables. Las variables serán utilizadas para indicar los datos ingresados en los campos de la aplicación, permitiendo al *script* ejecutar la aplicación con datos proveídos externamente durante su ejecución.
- **Prueba guiada por palabras claves (*Keyword-Driven Testing, KDT*):** Consiste en la utilización de tablas de datos y hacer una exploración en la misma de los *keywords*, que indican las acciones que se realizarán en la prueba automática.
- **Prueba guiada por comportamiento (*Behaviour-Driven Testing, BDT*):** Es un complemento del *Behaviour Driven Development, BDD*[6]. El objetivo de *BDT* es definir un lenguaje de negocio de dominio específico, que sea legible y permita describir el comportamiento de un sistema sin brindar detalles de cómo será implementado el comportamiento.

III. Definición del problema

El problema principal que este trabajo trata de resolver es la automatización de casos de prueba sobre las interfaces de aplicaciones web, evitando la programación de tales casos de pruebas (*scripts*) y adaptarlos a los cambios que surgen durante el desarrollo de una aplicación web y en tiempo de ejecución por la naturaleza dinámica en la generación de formularios (páginas), disminuyendo el tiempo y recursos para realizar su *testing* correspondiente.

A continuación los problemas específicos encontrados:

P1 Actualmente las herramientas de *testing* de quinta generación encontradas son comerciales, algunas ofrecen versiones de prueba o incompletas, en las cuáles las funcionalidades avanzadas están disponibles al adquirir el producto. Por otro lado, las herramientas *Open-source* brindan diferentes funcionalidades, sin embargo siguen utilizando *scripts* para la definición de casos de prueba.

P2 La definición y administración de los datos de pruebas para las interfaces de aplicaciones web puede resultar dificultoso, pues un ingeniero de *testing* deberá definir los datos para cada campo de cada formulario de la aplicación en prueba.

P3 Las herramientas *Open-source* actuales obtienen los datos a ingresar de los campos por medio de scripts generados a partir de herramientas de grabación de eventos del navegador, los valores están definidos en el código fuente de cada caso de prueba, o se utilizan valores aleatorios que muchas veces no representan buenos datos de prueba.

P4 Dependiendo del nivel II-D y enfoque II-C de *testing* aplicado, se podría contar o no con información de cómo funciona internamente una aplicación web. En muchos casos, no se cuenta con toda la información de la aplicación a probar, mas que su *URL* y datos de autenticación. Por otro lado, con un poco mas de información de cómo son manipulados los datos por el programador (durante el desarrollo de la aplicación), los datos de pruebas podrían ser mejores o mas enfocados a la lógica de negocio.

P5 Uno de los factores que pueden llegar a afectar negativamente la usabilidad de una aplicación web son los enlaces rotos. Un enlace roto es un enlace no disponible, ya sea porque el enlace ha dejado de existir, por generarse (dinámica o estáticamente) referencias de manera errónea, o por una variedad de motivos internos o externos.

P6 La mayoría de las herramientas de *testing Open-source* presentan reportes poco descriptivos de los resultados obtenidos durante la ejecución, un buen resumen de las ejecuciones facilitaría el trabajo de ingeniero de *testing* al momento de analizarlos.

P7 Existen pocas referencias de herramientas de *testing Open-source* que hayan sido utilizadas para pruebas de aplicaciones corporativas, no existen evidencias de su practicidad y uso en aplicaciones de tamaños importantes.

IV. Objetivos del trabajo

IV-A. Objetivos generales

El objetivo general es obtener una herramienta *Open Source* capaz de recorrer completamente una aplicación web, con la capacidad de generar automáticamente los datos de pruebas para los formularios desplegados, verificando a la vez su navegabilidad, proveyendo al ingeniero de *testing* una herramienta de automatización y gestión de *white box testing* y *black box testing* de manera sencilla y práctica.

IV-B. Objetivos particulares

Los objetivos particulares a cumplir en este trabajo son los siguientes:

O1 Contar con una herramienta Open-source, haciendo uso de herramientas existentes, extendiendo sus funcionalidades y combinando con otras tecnologías (Satisface P1).

O2 Diseñar un mecanismo para gestionar la información de los valores de prueba en los formularios web y detección de enlaces rotos (Satisface P2, P3 y P5).

O3 Proveer funcionalidades a la herramienta que permita aplicar los enfoques *Black Box Testing* y *White Box Testing* (Satisface P4).

O4 Generar reportes descriptivos en base a los resultados obtenidos en la ejecución de la herramienta de *testing* implementada (Satisface P6).

O5 Evaluar la herramienta propuesta y comparar los resultados obtenidos entre ambos enfoques implementados con aplicaciones web corporativas (Satisface P4 y P7).

[6]Es una metodología ágil de desarrollo que se centra en obtener una comprensión clara del comportamiento deseado del software a través de la discusión con las partes interesadas.

V. Modelo propuesto

Como propuesta de solución, se pretende diseñar e implementar una herramienta considerando las clasificaciones descritas en la sección II, como se cita a continuación:

- **Enfoque de *Testing* (II-C):** *White Box Testing* y *Black Box Testing*.
- **Nivel de *Testing* (II-D):** Prueba de Sistema.
- **Prueba Funcional (II-E):** Pruebas de Regresión.
- **Generación de Automatización de *Testing* (II-F1):** Quinta generación.
- **Enfoques de Automatización de *Testing* (II-F2):** Prueba de interfaz gráfica de usuario (GUIT) y prueba guiada por datos (DDT).

Para resolver los problemas planteados y lograr con los objetivos propuestos, se pretende realizar las siguientes tareas:

S1 Implementar una herramienta Open-source, en base a *frameworks* existentes. Debido a que *Crawljax* puede detectar eventos dinámicos (Ajax) e invocar automáticamente eventos web (*click*, *enter*, *submit*, etc.), se propone extender el *framework* de *Crawljax*, aprovechando esa característica y sus funcionalidades de reccorrido. Además, *Crawljax* cuenta con una interfaz web (*Crawljax Web UI Beta*) que facilita la ejecutación de pruebas y visualización de los resultados de las mismas (Satisface O1 y O4).

S2 Implementar una Base de Conocimiento para almacenar y gestionar la información de los valores de prueba en los formularios web y detección de enlaces rotos. Con esto se pretende tener un contenedor de datos de prueba que facilite el *testing* de aplicaciones, y otras directivas para el análisis de detección de enlaces rotos (Satisface O2).

S3 Implementantar un componente como *plugin* para obtener automáticamente posibles valores de prueba para autocompletar los campos de los formularios web en las pruebas de interfaz, a partir de la Base de Conocimiento (Satisface O2).

S4 Desarrollar las funcionalidades necesarias para aplicar *White Box Testing*. Si se cuenta con el código fuente de la aplicación web, obtener información que facilite la manera en que serán verificados los formularios de las páginas web, por medio de meta-datos definidos en el *HTML* de la aplicación en prueba (Satisface O3).

S5 Desarrollar las funcionalidades necesarias para aplicar *Black Box Testing* en caso de no contar con acceso al código fuente de la aplicación web, inferiendo los datos de cada campo encontrado (Satisface O3).

S6 Implementantar un componente como *plugin* para la detección de enlaces rotos en una aplicación web, a partir de la Base de Conocimiento. (Satisface O2).

S7 Implementantar un componente como *plugin* para generar reportes de los datos obtenidos en la ejecución de la herramienta (Satisface O4).

S8 Ejecutar la herramienta con aplicaciones web desarrolladas por la FPUNA, comparando los resultados obtenidos para cada enfoque de *testing* de implementado (Satisface O5).

Se describe el modelo de interacción entre los componentes mencionados. En la Figura 1 se puede apreciar un esquema de los componentes de la herramienta a implementar:

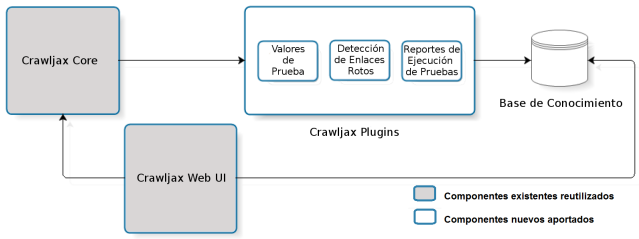

Figura 1. Modelo propuesto.

- **Crawljax:** Los componentes ya existentes a ser utilizados en este modelo son los siguientes:
  - **Crawljax Web UI:** Al momento de la redacción de esta propuesta, la versión de está plataforma es Beta (3.5.1). Se pretende aprovechar esta interfaz para facilitar al ingeniero de *terting* la configuración de Crawljax.
  - **Crawljax Core:** Herramienta que permite realizar un recorrido completo de una aplicación web, sus funcionalidades son extensibles con la implementación de *plugins*,
- **Plugins:** Los siguientes *plugins* a ser implementados proveerán las nuevas funcionalidades para la automatización del *testing*.
  - **Valores de Prueba:** Obtendrá automáticamente los valores registrados en la base de conocimiento y los utilizará para cargar los campos de los formularios web.
  - **Detección de Enlaces Rotos:** Detectará enlaces rotos si existieran durante el recorrido, registrándolos para luego ser verificados.
  - **Reporte de Ejecución de Prueba:** En este módulo se implementarán los reportes con los resultados obtenidos por las nuevas funcionalidades implementadas.
- **Base de conocimiento:** Basado el enfoque de *Data-Driven Testing* II-F2, este componente almacenará los valores de prueba e informaciones para detectar enlaces no encontrados durante la navegación, a ser utilizados por los plugins implementados.

Referencias

[1] D. H. W. y W. G. Macready, "Ecmascript language specification," ECMA International, Tech. Rep. Standard ECMA-262, 2011.

[2] J. J. Garrett, "Ajax: A new approach to web applications," http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications/.

[3] C. Kaner, "Exploratory testing," in *Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference*, Orlando FL, 2006.

[4] IEEE, "Ieee standards collection: Glossary of software engineering terminology," in *IEEE Standard 610.12-1990*, 1990.

[5] J. Pan, "Software testing," Carnegie Mellon University, Tech. Rep. 18-849b, 1999.

[6] IEEE, "Ieee standard glossary of software engineering terminology," in *IEEE Standard 610.12-1990*, 1990.

[7] B. B. y Victor R. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, pp. 135–137, January 2001.

[8] C. Nagle, "Test automation frameworks," Web.

[9] J. H. y Martin Gijsen, "Fifth generation scriptless and advanced test automation technologies."