Testing en aplicaciones web

Ariel Pasini[1], Hugo Ramon[2], Pablo Thomas[2]
{apasini,hramon,pthomas}@lidi.info.unlp.edu.ar
III-LIDI [3] Facultad de Informática UNLP

Resumen

Generalmente las técnicas de testeo convencionales conducen a encontrar errores en sistemas de escritorio estándar. El comportamiento y características especiales de una aplicación web generan errores no detectados por estas técnicas, demandando un tratamiento especial, particularmente en lo referido a la navegación. Considerando a este tipo de aplicaciones como máquinas de estado finito se obtiene un diagrama navegacional, del cual se deriva un árbol de testeo que conduce a la generación del conjunto de casos de testing. Se presenta la aplicación de este esquema en un caso real, donde los resultados obtenidos ratifican la importancia de realizar testing desde el punto de vista navegacional.

Palabras Claves
Aplicaciones Web - Diagrama Navegacional – Arbol de Testing – Casos de Testeo

1. Introducción

Existen varias definiciones formales de calidad en software, pero una definición sencilla podría ser que los usuarios estén satisfechos de la aplicación desarrollada. Si se considera la calidad como un proceso que atraviesa todas las etapas del ciclo de vida de desarrollo del software, este proceso no implica solo cumplir satisfactoriamente cada etapa, sino optimizar cada una de ellas logrando una mejora continua. En este contexto, la etapa de testing es fundamental para asegurar la calidad final del producto de software [1].

Durante la generación de software generalmente se producen errores. No todos ellos son fáciles de detectar por el desarrollador que realiza la tarea. Para encontrar esos errores se utilizan las *técnicas de testing*, que generalmente se clasifican en tres grandes grupos: funcionales, estructurales y basadas en errores [1].

Las funcionales consisten en buscar y ejecutar los casos de testeo, observando sólo aspectos funcionales y los requisitos deseados para ellos. Los criterios mas relevantes utilizando esta técnica son “partición en clases”, “análisis de valores límites” y “grafo de causa efecto”. Las estructurales consisten en analizar las líneas de código de un programa evaluando los cambios de flujo del programa. Se destacan los criterios basados en “flujos de control”, “flujos de datos”, y “complejidad”. Por ultimo, las técnicas basadas en errores, utilizan los errores cometidos por los programadores que no impiden la ejecución del sistema pero si alteran el resultado. El criterio más relevante de esta técnica es el de “análisis mutante”.

Actualmente, existe gran demanda de aplicaciones Web con tiempo acotado para su desarrollo. Esto produce una disminución de los controles de calidad, con efectos secundarios muy costosos. Hay requerimientos no funcionales [2] como la confiabilidad, funcionalidad, seguridad, disponibilidad, escalabilidad, entre otros, que no son posibles omitir ya que esta falta conduciría directamente al fracaso del proyecto.

1 JTP. Dedicación Exclusiva. Facultad de Informática. UNLP
2 Profesor Adjunto. Dedicación Exclusiva. Facultad de Informática. UNLP
3 Instituto de Investigación en Informática. Calle 50 y 115 1º Piso La Plata .Facultad de Informatica. UNLP

En este trabajo se analiza la forma de realizar testeo sobre una aplicación real basadas en la Web desde una óptica navegacional, utilizando DNP (Diagrama de Navegación de Páginas)

El trabajo consta de siete secciones, siendo la primera la introducción. En la segunda sección se describen las técnicas de testing más relevantes, en la tercera se puntualiza el testeo de aplicaciones Web. Posteriormente se describe el modelo de comportamiento de una aplicación web, con el análisis de un caso específico. Finalmente, se plantean las conclusiones obtenidas y se proponen trabajos futuros.

2. Testing

Habitualmente se tiende a definir testing/testeo (de aquí en más tratados como sinónimos) como "probará en ejecución el programa para ver si funciona correctamente", o "es un proceso que lleva a confiar que el programa hará lo que se supone que debe hacer". Pero esto es un grave error, ya que realizar testing implica exactamente lo contrario, testear es el "proceso de ejecutar un programa con el fin de encontrar errores" [3].

Ligado a la prueba de un programa se encuentran los casos de testing. Estos casos son los conjuntos de pares de valores (a,b) donde "a" es el valor ingresado, y "b" es el valor esperado para la ejecución de "a". Habitualmente se denota que un caso de testeo fue exitoso cuando no encuentra errores. Analizando la definición de testeo con precisión se comprueba que se trata de lo opuesto: *un testeo será exitoso cuando detecte un error*.

La comunidad de Ingeniería de Software clasifica los testing en tres grandes grupos: "Testeo Funcional o Caja Negra", "Testeo Estructural o Caja Blanca" y "Testeo basados en errores"

2.1 Testeo Funcional (Caja negra)

Esta técnica se utiliza para verificar las funcionalidades del sistema sin analizar su codificación. El proceso aplicado consiste en identificar la funcionalidad que posee un sistema, y luego crear casos de testeo capaces de evaluar si el software satisface la funcionalidad esperada [4][5].

Los criterios de testeo funcionales mas relevantes son:

- **Partición en Clases de Equivalencia:** Se divide el dominio de entrada en clases de equivalencia, luego se forman los casos de testeo con un dato representativo de cada clase.
- **Análisis de Valores Límites:** Es un complemento del partición en clases de equivalencia, analizando rigurosamente los límites de cada clase. En este caso, además de tomar cualquier dato como representativo, se toman los datos de los límites de la clase.
- **Grafo de Causa Efecto:** Los criterios anteriores no exploran la combinación de los datos de entrada. Este criterio se basa en esa omisión, tomando las posibles condiciones de entrada (causa) y las acciones posibles (efectos) del programa. Posteriormente se construye un grafo y se lo convierte en una tabla de decisión desde donde se derivan los casos de testeo.

2.2 Testeo Estructural (Caja Blanca)

Esta técnica se basa en el conocimiento de la estructura interna del programa. Se representa el programa como un grafo, donde cada instrucción de flujo de control -genera un arco hacia un nuevo nodo, para luego determinar los requisitos del testeo [4] [5]

Los criterios estructurales se clasifican en:

- **Criterios Basados en flujos de control**: Utilizan el control de ejecución de programa. Los criterios mas conocidos de esta clase son:
  - "Todos los nodos", que exige que cada vértice del grafo sea ejecutado al menos una vez,
  - "Todos los arcos", que exige que todos los arcos (es decir, cada desvío de control de flujo) sean ejecutado al menos una vez,
  - "Todos los caminos", que exige que todos los caminos posibles sean ejecutados.
- **Criterios Basados en flujo de datos:** Utiliza información de flujo de datos para determinar los requisitos del testeo. Este criterio explora las interacciones que envuelven las definiciones de las variables y las referencias a las mismas.
- **Criterios basados en complejidad:** Utiliza información sobre la complejidad del programa para derivar los requisitos del testeo. Implica que un conjunto de caminos linealmente independientes del grafo sea ejecutado.

2.3 Testeo basado en errores

Esta metodología utiliza información sobre los tipos de errores mas frecuentes del proceso de desarrollo de software. Los casos de testeo generados son específicos para mostrar la presencia o ausencia de errores. [4][6]

El criterio típico de esta técnica es Análisis Mutante, donde a un programa P se le realizan pequeñas alteraciones sintácticas generando una serie de programas P1, P2…Pn denominados Mutantes. El objetivo es determinar un conjunto de casos de testeo que consiga revelar, a través de la ejecución del programa, las diferencias de comportamiento existentes entre el programa y sus mutantes [4].

3. Testeo web

El auge de las aplicaciones Web en el mercado de software ha propiciado el desarrollo de sistemas utilizando metodologías convencionales, sin considerar los nuevos problemas que este tipo de soluciones traen aparejadas.

El comercio electrónico, las aplicaciones distribuidas, el trabajo colaborativo, entre otras, son actividades comunes en el desarrollo de software convencionales y muy factibles de migrar a aplicaciones basadas en la Web. Pero si esta tarea es realizada de forma directa, sin tener en cuenta las diferencias existentes, no tendrá un éxito. Si se considera un sistema de escritorio tradicional y se migra a un entorno Web, se presentarán varias diferencias: [7] [8] [9]

- **Clientes dinámicos:** Ya no se tiene un cliente de forma estática, sino que el servidor va a ir generando dinámicamente las interfases del cliente para cada petición.
- **Clientes desde Sistemas Operativos diferentes:** Es posible que los clientes se conecten desde sistemas operativos diferentes, por lo que las interfaces deben funcionar correctamente en cada uno de ellos.
- **Diferentes tipos de conexión:** Los clientes pueden conectarse a través de diferentes tipos de redes. Los puntos de conexión pueden variar de velocidades, tiempo de respuesta, protocolos de transmisión, etc.
- **Alteraciones del control de flujo por parte del usuario:** En una aplicación convencional los usuarios no pueden alterar el flujo del programa, mientras que en las aplicaciones Web los usuarios pueden alterar el flujo de control presionando las teclas de

retroceso o actualización de página, cambiando el contexto de ejecución y generando efectos inesperados. Estas posibilidades deben estar contempladas en el desarrollo de una aplicación Web.

- **Cambios de configuración:** El usuario puede cambiar la configuración del cliente (como por ejemplo deshabilitando cookies), produciendo así un cambio de comportamiento en la aplicación.
- **Problemas en la programación:** Existen características particulares en el desarrollo de sistemas orientados a la web, que generan una necesidad de mayor testeo que en las aplicaciones de escritorio estándar. Algunas de esas características son la utilización de varios lenguajes para su codificación (entre otros, HTML o Javascript en el cliente y Java o Perl para ser ejecutado en el servidor), el alto reuso de código existente (sin implicar que este código ya esté testeado) y la utilización de componentes de terceros (que no siempre son garantía de calidad).
- **Problemas de interacción:** Los sistemas distribuidos basados en la Web se caracterizan por utilizar o brindar servicios de otros sitios. Tal es el caso de los sistemas que admiten pagos con tarjetas de crédito, en los cuales se interactúa con un sistema bancario que brinde efectivamente el servicio de cobro con tarjeta.

  La información que utiliza un sistema Web puede estar distribuida en diferentes bases de datos ubicadas en diversos puntos geográficos.

  Además, la usabilidad característica de estos sistemas, hace que personas con diversas aptitudes sean usuarios potenciales.

  Las particularidades enunciadas generan un problema inherente en la interacción de los sistemas web con otros sistemas, con las bases de datos manipuladas y con la gran diversidad de usuarios hipotéticos existentes, incrementándose la necesidad de testeo de estas aplicaciones.
- **Mantenimiento:** Las tecnologías utilizadas en las aplicaciones basadas en la Web avanzan muy rápido, lo que requiere que el mantenimiento se deba llevar a cabo mas frecuentemente y de forma mas eficiente, especialmente en lo que respecta a seguridad.

4. Modelo de comportamiento de una aplicación web

Es posible representar el modelo de comportamiento de una aplicación web, a través de la navegación de las páginas de la aplicación.

El DNP es una maquina de estados finita [10], donde cada estado representa una página y las transiciones entre los estados representan links que para identificarlos se los etiqueta con sus respectivas URL. Las guardas indican las condiciones necesarias para realizar los pedidos. El diagrama representa todas las páginas generadas por los pedidos realizados a los servicios web como estados de la aplicación.

Para detectar errores relacionados con la navegación se construye el árbol de testing desde el DNP [11]. Los nodos del árbol representan los estados (paginas) y las aristas las transiciones (link o requerimientos).

El árbol de testing se utilizará para generar el conjunto de casos de testeo. Cada uno de esos casos es una secuencia de nodos desde la raíz hasta una hoja. El árbol permite entonces controlar la accesibilidad de las páginas y la existencia de deadlock en la aplicación.

En le caso que se direccione una URL externa, se genera una hoja del árbol, dado que al ser un enlace externo, no se tiene la posibilidad de modificar las actividades que surgen a partir de esa URL externa.

El Instituto de Investigación en Informática III-LIDI ha desarrollado un Sistema orientado a la Web para un Consorcio de Inmobiliarias de España. Esta aplicación, actualmente en producción, incluye las operaciones transaccionales clásicas de altas, bajas y modificaciones, y la posibilidad de realizar un análisis comparativo de mercado [12]. Esta funcionalidad, compleja para cualquier aplicación de escritorio, lo es aún más para un sistema web, dado el nivel de interactividad requerido. Además es particularmente importante destacar que para este Proyecto se utilizó un framework de desarrollo denominado “PHP4DB” [13] que permite automatizar tareas rutinarias de codificación en un ambiente LAMP (Linux + Apache + MySQL + PHP) basado en el modelo relacional clásico. [14].

La implementación remota y las características propias de esta aplicación, generó la necesidad de testing, que en este caso fue exhaustivo.

En la **Figura 1** se representa el modelo de comportamiento navegacional parcial, de la página Lista de Propiedades, detallando la operación “Agregar Propiedad”.

La navegación debería respetar un comportamiento hipotético definido. Los pasos a seguir para respetar este comportamiento serían:

1) Desde la pagina de *Login* se ingresan los datos de usuarios y se envían al servidor a través de la urlLogin
2) Como respuesta, el servidor retorna dos posibles opciones OK o error.
3) En el caso de error se deberá visualizar una *pagina de error* y permitir volver al *Login*.
4) Si la respuesta fue afirmativa se accede al menú de opciones donde se seleccionan Propiedades, que a través de la urlPropiedades conducirá a una pagina con la *lista de propiedades* donde es posible, ordenar, aplicar filtros, navegar hacia adelante y atrás en la lista visualizada. Estas operaciones requieren de consultas en el servidor por lo que se debe contemplar que las acciones pueden retornar un error, en cuyo caso deberá ser informado y permitir volver al estado anterior.

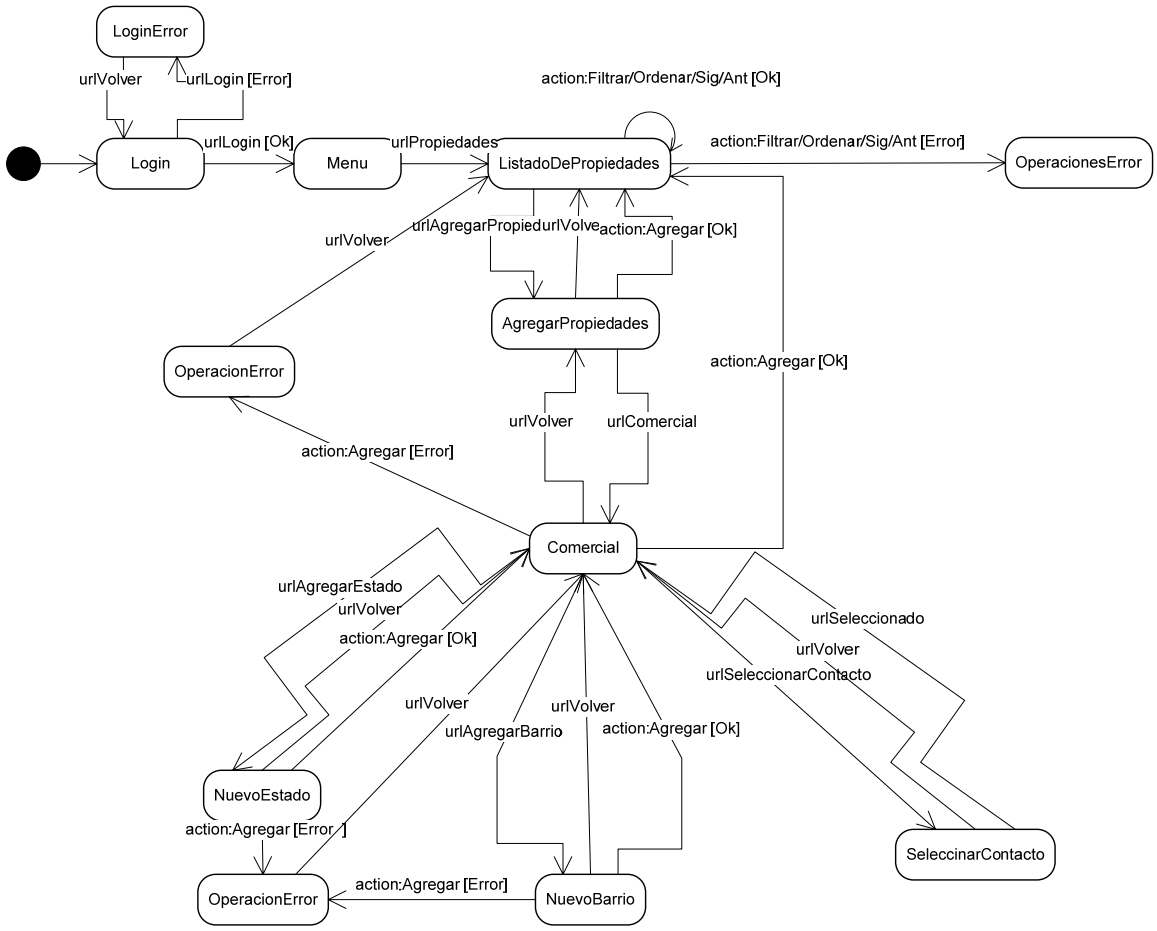

Figura 1 DNP Operación Agregar Propiedad

5) La pagina de *lista de propiedades* entre otras opciones (no contempladas en el diagrama) tiene la de *agregar propiedad*, que, si es seleccionada, permitirá optar entre los tipos de propiedad disponible, analizando en este caso el tipo *comercial*. Desde allí además de los datos simples a completar, se encuentran Barrio y Estado que deben ser seleccionados y/o agregados en el caso de no existir.
6) Luego se debe regresar a la página en cuestión luego de haber realizado la operación con éxito o haber cancelado la operación.
7) En el caso de los contactos se selecciona de una pagina y se retorna habiendo realizado o no una selección.
8) Todas las operaciones de alta pueden poseer una respuesta desfavorable desde el servidor, cuyo mensaje de ser manejado y permitir regresar a la instancia que corresponda.

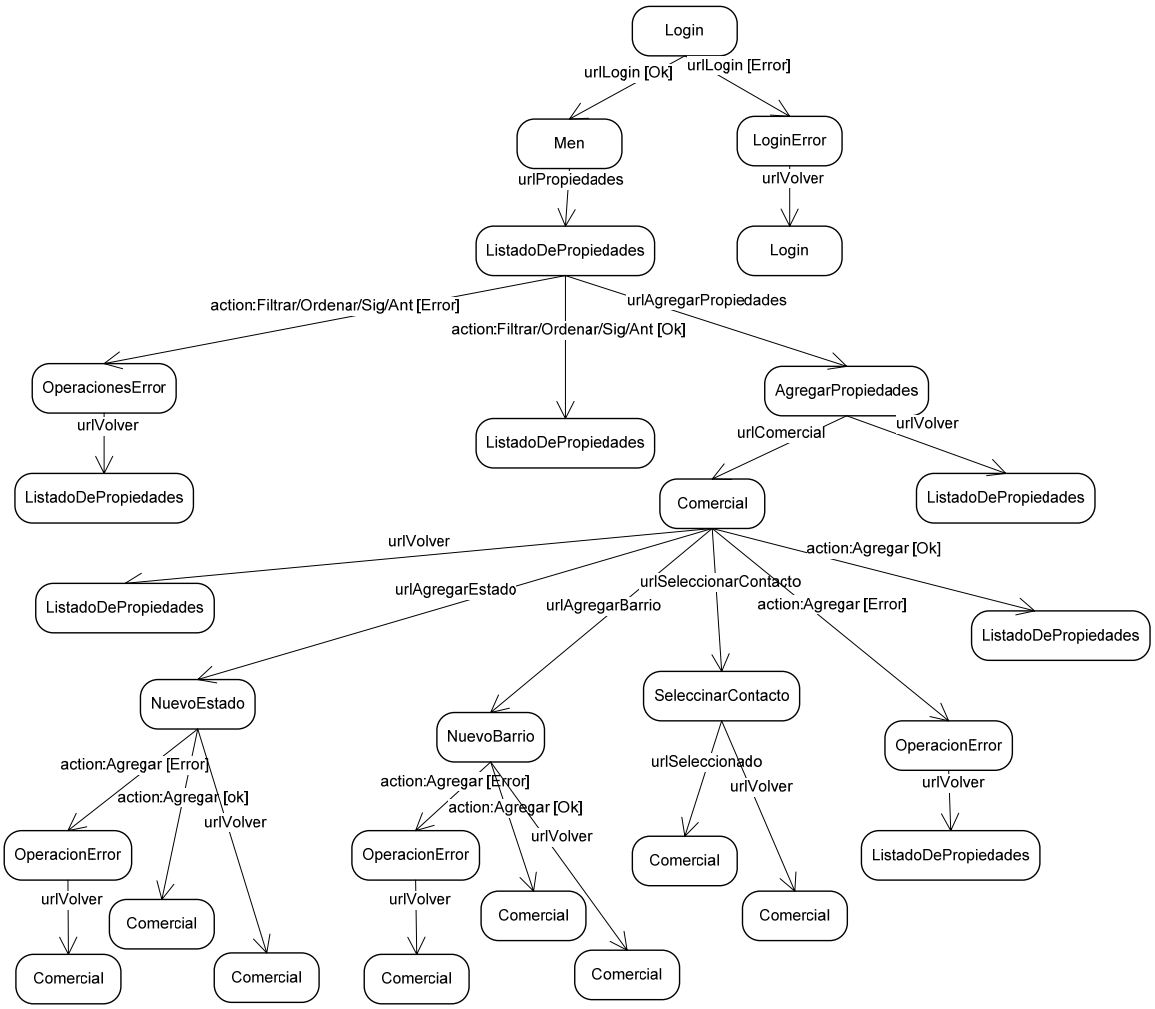

Figura 2 Árbol de Testeo

A partir del diagrama de navegación comenzando en el estado inicial, es posible generar el árbol de testeo recorriendo todas las salidas de cada uno de los estados (páginas) hasta finalizar cada operación, o directamente regresar al estado de origen.

El árbol es representado en la **Figura 2** donde se puede realizar el recorrido mencionado y generar los casos de testing. Como se puede comprobar, cada rama de este árbol es un caso de testo en si mismo.

Un modelo de representación posible de cada uno de estos casos es el generado en la **Tabla 1**. En esta tabla se definen las URL´s a navegar con el respectivo resultado esperado. Para reducir la longitud de nombres sólo se utilizaron las iniciales de cada estado (ej L: Login, LDP: Listado de Propiedades, C: Comercial)

	Caso de Testing	**Resultado Esperado**
1	L.urlLogin[Error]/LE.urlVolver/L	L
2	L.urlLogin[Ok]/M.urlPropiedades/LDP.action:Filtrar[Error]/OE.urlVolver/LDP	LDP
3	L.urlLogin[Ok]/M.urlPropiedades/LDP.action:Filtrar[Ok]/ LDP	LDP

4	L.urlLogin[Ok]/M.urlPropiedades/LDP.action:Ordenar[Error]/OE.urlVolver/LDP	LDP
5	L.urlLogin[Ok]/M.urlPropiedades/LDP.action:Ordenar[Ok]/ LDP	LDP
6	L.urlLogin[Ok]/M.urlPropiedades/LDP.action:Sig[Error]/OE.urlVolver/LDP	LDP
7	L.urlLogin[Ok]/M.urlPropiedades/LDP.action:Sig[Ok]/ LDP	LDP
8	L.urlLogin[Ok]/M.urlPropiedades/LDP.action.Ant[Error]/OE.urlVolver/LDP	LDP
9	L.urlLogin[Ok]/M.urlPropiedades/LDP.action.Ant[Ok]/ LDP	LDP
10	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url Volver/LDP	LDP
11	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url AgregarEstado/AE.action:Agregar[Ok]/C	C
12	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url AgregarEstado/AE.action:Agregar[Error]/ OE.urlVolver/C	C
13	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url AgregarEstado/AE.urlVolver/C	C
14	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url AgregarBarrio/AB.action:Agregar[Ok]/C	C
15	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url AgregarBarrio/AB.action:Agregar[Error]/ OE.urlVolver/C	C
16	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.url AgregarBarrio/AB.urlVolver/C	C
17	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.urlSeleccionar/SC.urlSeleccionado/C	C
18	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.urlSeleccionar/SC.urlVolver/C	C
19	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.action:Agregar[Ok]/LDP	LDP
20	L.urlLogin[Ok]/M.urlPropiedades/LDP.urlAgregarPropiedades/AP.urlComecial/C.action:Agregar[Error]/ OE.urlVolver/LDP	LDP

Tabla 1 Casos de Testing

El resultado de ejecutar los casos definidos en la tabla, mostró la existencia de siete éxitos del testeo (errores de la aplicación) y trece fracasos (ausencia de errores de la aplicación). Este resultado permitió corregir los errores encontrados. Además se comprobó ausencia de páginas inaccesibles y de deadlock, ya que el árbol permitió retornar siempre a páginas con salida.

5. Conclusiones

El diagrama navegacional de todas las operaciones del subsistema de Propiedades permitió generar el árbol de casos de testeo, a partir del cual se obtuvieron 58 casos de testeo. La ejecución de esos casos permitió detectar 24 errores de navegabilidad, correspondientes en su gran mayoría a la ausencia de páginas de manejo de excepciones por parte del servidor.

Los resultados mostrados hasta aquí confirman la utilidad de aplicar un modelo de testeo a un sistema orientado a la web con dos beneficios esenciales. En primer lugar, se corrigieron errores no detectados hasta ese momento, optimizando así la calidad del sistema. En segunda instancia esta

información fue utilizada como retroalimentación para el framework de desarrollo PHP4DB, de modo de poder evitar estos errores en aplicaciones futuras generadas con esta herramienta.

6. Trabajos Futuros

Considerando que una aplicación Web se puede analizar como una maquina de estado finita, se aplicará la técnica de testeo basada en errores con el criterio de Análisis Mutante [15]. Por otro lado se continuará optimizando el framework de desarrollo PHP4DB en base a la retroalimentación generada por los proyectos, comparando resultados (de desarrollo y testing) con otros métodos de producción de software web como OOWS (Object Oriented Web)[16], WEBML [17] y OOHDM [18].

7. Referencias

[1] Pasini Ariel, Vergilio Silvia, **"Testeo de Aplicaciones Web"**, informe técnico, Programa UNLP - Universidad Federal do Paraná, Curitiba, Brasil, Abril 2006.

[2] Cysneiros, L.M., Leite, J.C., "**Using Non-Functional Requirements to Improve the Software Development Process"**, Notas de curso tutorial, ICRE, Chicago

[3] Myers G., **"El arte de probar software",** Editorial El Ateneo, 1983.[4] Ellen Barbosa et.al., **"Introducción a los casos de test de Software"**, Notas Didácticas. Instituto de Ciencias Matemáticas de San Carlos - USP – San Carlos. ISSN 010325770, 1998

[5] Roger S. Pressman: **"Ingeniería del Software"**, Editorial Mc Graw Hill, 5º Edición.

[6] DeMillo, R. A. and Lipton, R. J. and Sayward, F. G., **"Hints on Test Data Selection: Help for the Practicing Programmer"**. IEEE Computer Vol.: 11 Nro.: 4: 34-41, Abril 1978.

[7] Offutt J., **"Quality Attributes of Web Software Applications"**. IEEE Software: Special, Issue on Software Engineering of Internet Software 19 (2):25-32, Marzo/ Abril 2002.

[8] Wu, Y. y Offutt, J., **"Modeling and testing web-based Applications"**, https://citeseer.ist.psu.edu/551504.html: 1-12, Julio 2004

[9] Offutt J., **"Web Software Applications Quality Attributes"**. *Quality Engineering in Software Technology (CONQUEST 2002)*, pages 187-198, Nuremberg, Germany, September 2002.

[10] Kung et. al., **"An Object-Oriented web test model for testing web application"**. IEEE Press. The 24th Annual International Computer Software and Applications Conference, COMPSAC 2000:537-542, 2000.

[11] Kung et. al., **"An Object-Oriented web test model for testing web application",** IEEE. Quality Software First Asia-Pacific Conference: 111-120, October 2000.

[12] http://homebuying.about.com/library/glossary/bldef4.htm

[13] Delia L., Caseres G., Ramon H., Thomas P, **"Herramienta Ágil para Desarrollo WEB"**, Informe Técnico III-LIDI Nº 02-05-001/06, Mayo 2006.

[14] M. Torchiano, M. Morisio, **"Overlooked Aspects of COTS-Bases Development"**, IEEE Software, 2004.

[15] Sandra Pinto, Ferraz Fabri et. al., **"Mutation Analysis Testing for Finite State Nmachines"**, IEEE software 220-229, 1994.

[16] Fons, Pastor, Valderas, Ruiz, **"OOWS: Un Método de Producción de Software en Ambientes WEB"**, capitulo 9 del libro **"Avances en Comercio Electrónico"**, ISBN 84-607-5827-3, 2002.

[17] Ceri S., Fraternali P., Bongio **“A. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites”**. In WWW9, Vol. 33 (1-6), pp 137-157. Computer Networks, 2000.

[18] Schwabe D., Rossi G. and Barbosa D.J., **“Systematic Hypermedia Application Design with OOHDM”**. Proc. ACM Conference on Hypertext. pp.166. 1996.