Kobold: Usabilidad como Servicio a través de Refactorings de Interacción Client-Side

» Julián Grigera
LIFIA, Facultad de Informática, Universidad Nacional de La Plata / CONICET / CICPBA, Argentina

Alejandra Garrido, Juan Cruz Gardey y Gustavo Rossi
LIFIA, Facultad de Informática, Universidad Nacional de La Plata / CONICET, Argentina

Resumen

Las aplicaciones web se han convertido en herramientas fundamentales para nuestras tareas cotidianas, negocios, interacción social e intercambio de información en general, pero la mala usabilidad continúa siendo un problema frecuente en ellas. Muchos de estos problemas han sido catalogados, pero su evaluación sistemática y reparación siguen siendo costosos. Tanto desde la academia como desde la industria, han surgido esfuerzos para automatizar los tests de usabilidad o presentar estadísticas, pero no suelen considerar las posibles soluciones para los problemas hallados. En los casos en que se muestran soluciones, suelen ser en forma de guías o patrones que se pueden aplicar manualmente. En este trabajo presentamos Kobold, un servicio que detecta problemas de usabilidad a partir de la captura de eventos de interacción, y ofrece soluciones para repararlos automáticamente cuando sea posible, o al menos sugerir soluciones concretas. Además, permite la generación de versiones diferentes de la interfaz web, cada una con un conjunto de *refactorings*, para poder realizar pruebas antes de aplicar los cambios, aprovechando la infraestructura de producción de la aplicación analizada sin necesidad de crear ambientes de prueba. Kobold utiliza la técnica de *refactoring* y el concepto de *bad smell*s, lo que facilita el reconocimiento de los problemas y soluciones incluso para aquellos con poco conocimiento en usabilidad.

KEYWORDS: USABILIDAD WEB, USABILITY REFACTORING, SOFTWARE AS A SERVICE.

1. Introducción

Los problemas de usabilidad son comunes en las aplicaciones web. Los efectos de estos problemas van desde generar frustración en los usuarios y el impedimento de realizar tareas básicas, hasta un impacto en la facturación en los casos de las aplicaciones comerciales. Una manera popular para asesorar los problemas de usabilidad es realizar tests de usuario (Rubin

and Chisnell, 2008), que brinda la ventaja de observar usuarios reales de forma directa. Sin embargo, esto requiere de tiempo y de recursos considerables, además de expertos capacitados para analizar los resultados y descubrir los problemas, y en algunos casos, proponer las soluciones correspondientes. Por este motivo, muchas organizaciones prefieren priorizar el uso de recursos en otros aspectos del desarrollo. Existen herramientas que permiten automatizar la captura de eventos relacionados con la interacción, pero éstas suelen estar enfocadas en brindar estadísticas o datos agregados que requieren de la interpretación humana, y también suelen carecer de propuestas o sugerencias para solucionar los problemas, excepto en casos puntuales como tiempos de carga. Las sugerencias para mejorar usabilidad suelen estar publicadas como guías o patrones para aplicar durante el desarrollo, pero esto, además de requerir intervención manual, es más difícil de aprovechar en los casos en que la aplicación ya se encuentra en producción. Además, si bien las guías y patrones son generalmente creados por expertos en el área, no puede asegurarse su éxito en todos los casos, por lo cual se necesitan tests adicionales para comprobar su adecuación antes de aplicarlos.

Nuestra propuesta para abordar los problemas recién mencionados consiste en brindar un medio para mejorar la usabilidad web en aplicaciones productivas, utilizando la mínima cantidad de recursos posible, y haciéndola disponible para el público general, en particular para aquellos sin conocimiento sobre usabilidad en general. El objetivo es proveer un servicio automatizado para descubrir los problemas con los cuales se topan los usuarios finales mientras interactúan con la aplicación web, y sugerir o incluso aplicar soluciones concretas para estos problemas siempre que sea posible. Además, debe ser posible validar las soluciones antes de aplicarlas para el público general, dado que los desarrolladores suelen ser (naturalmente) reticentes a agregar código ajeno a una aplicación en producción, por cuestiones de seguridad o compatibilidad.

Para caracterizar y resolver los problemas de usabilidad, se utiliza la noción de *bad smells* y *refactorings*. Esta técnica, originalmente utilizada para mejorar la calidad interna del código fuente, fue llevada al área de usabilidad (Garrido, Rossi and Distante, 2011). La idea consiste en mejorar incrementalmente la usabilidad web, definiendo *usability refactorings* que realizan cambios simples en la navegación, presentación o interacción en general, preservando la funcionalidad original – de la misma manera en la que los refactorings de código preservan el comportamiento. En particular, los *refactorings* que se proponen en este trabajo son aplicados del lado del cliente (CSWR por sus siglas en inglés: *Client-Side Web Refactorings*), lo que brinda como ventaja que no sea necesario modificar el código fuente de la aplicación a refactorizar, y permite llevar a cabo la propuesta en forma de servicio. Un ejemplo de CSWR sería "*Add Client Validation*", que dado un formulario que realiza validaciones del lado del servidor, se le agregan validaciones (por ejemplo, de un campo requerido) del lado del cliente.

Dado que un mismo problema de usabilidad en diferentes contextos puede requerir diferentes soluciones, la aplicación de un CSWR no siempre garantizará una mejora de usabilidad. Por este motivo es importante proveer una manera de realizar pruebas en un ambiente seguro antes de aplicar definitivamente un refactoring. Puesto que la presente propuesta tiene como objetivo demandar la menor cantidad posible de recursos, esto no debería representar una inversión adicional en infraestructura.

Para implementar estas ideas, presentamos Kobold, una herramienta utilizable tanto por desarrolladores como expertos en usabilidad (Grigera, J., Garrido, A., & Rossi, G. 2017). Kobold

permite brindar asesoramiento sobre usabilidad en aplicaciones que ya se encuentran corriendo en producción, y provee sugerencias de soluciones que, en muchos casos pueden aplicarse automáticamente.

2. Trabajo relacionado

El presente trabajo está basado en la idea de capturar automáticamente la interacción de los usuarios finales para detectar defectos de usabilidad. Esta idea ha sido empleada antes, incluso fuera del ámbito de la web. Uno de los primeros investigadores en aplicarla en 2002 fue Chi, que propuso sofisticadas técnicas de visualización para ayudar a los expertos a comprender los grandes volúmenes de datos generados por los usuarios (Chi, 2002). Su método de visualización incluye una estructura basada en uso, patrones de tráfico y predicción de caminos comparados con caminos reales. Hoy en día, los servicios de analítica web provistos por herramientas comerciales como CrazyEgg[1], Hotjar[2], Mouseflow[3] o Woopra[4] capturan automáticamente la interacción de los sitios en producción, ofreciendo mapas de calor y otras estadísticas que muestran visualmente el comportamiento de los usuarios (la Fig. 1 muestra un mapa de calor –*heatmap*- típico de estas herramientas).

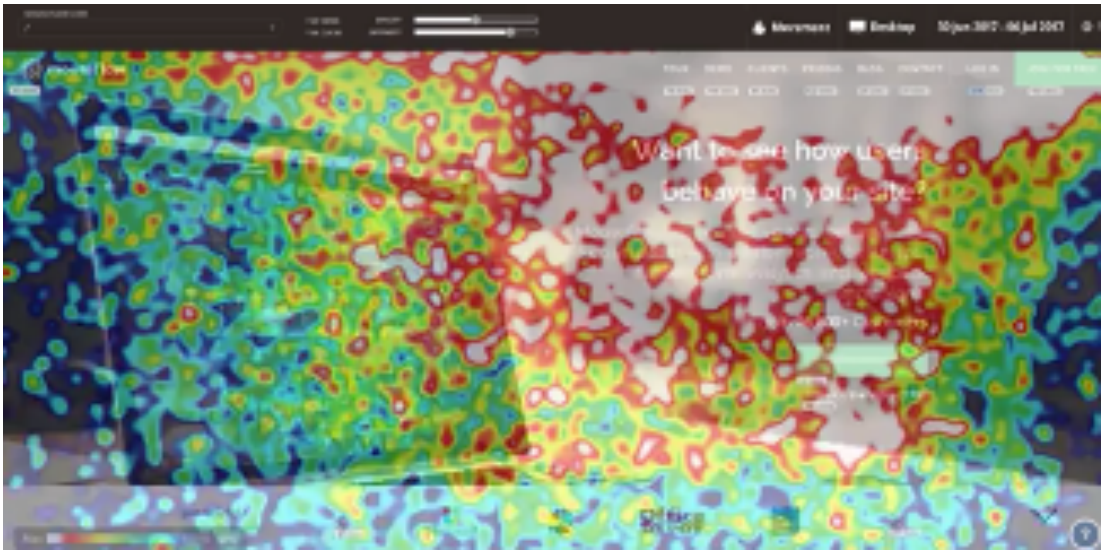

Fig. 1. Ejemplo de un mapa de calor que grafica la frecuencia de interacción de los usuarios por zonas de la página.

Estos servicios demandan menos recursos, ya que los usuarios finales son quienes, sin saberlo, proveen los datos necesarios. Sin embargo, los datos estadísticos requieren de un experto en usabilidad para descubrirlos y hallarles soluciones.

En un trabajo de 2005, Saadawi *et al*. (2005) destacaron que los tests manuales tenían un alto costo en tiempo y recursos, y desarrollaron un sistema de captura de logs de interacción

1 Crazy Egg www.crazyegg.com
2 Hotjar www.hotjar.com
3 MouseFlow www.mouseflow.com
4 Woopra www.woopra.com

en sistemas desktop. En su trabajo, los autores concluyen que, para un universo específico de problemas de usabilidad, la herramienta automática es capaz de capturar los mismos errores que un test *think aloud*. La estructura del experimento en este trabajo es la misma que se realizó en la presente propuesta para comprobar la precisión de la captura automática de problemas de usabilidad.

En trabajos más recientes sobre el estudio de logs de interacción, la propuesta de Apaolaza, Harper y Jay (2015) analiza registros de eventos de bajo nivel "en el llano", es decir tomándolos de usuarios reales realizando tareas libremente en sitios reales. Ellos transforman estos logs en entidades de más alto nivel llamadas *micro-comportamientos*, y las utilizan para demostrar cómo el comportamiento de los usuarios evoluciona con el tiempo. En un trabajo más reciente, Apaolaza *et al*. presentan WevQuery (Apaolaza y Vigo, 2017), una herramienta que permite realizar análisis sofisticado de eventos de bajo nivel mediante la generación de consultas (*queries*) para hallar secuencias específicas. Las consultas se generan con un lenguaje visual y permiten realizar análisis en grandes volúmenes de logs de interacción previamente generados. A diferencia de lo que se propone en este trabajo (reportar problemas concretos para una audiencia amplia), queda en los expertos la tarea de exploración de los eventos mediante consultas que, si bien constituyen una manera muy potente y flexible de observar la interacción, requiere de una mano experta.

La búsqueda de problemas de usabilidad web en ambientes no controlados requiere diferentes técnicas de análisis debido a que no hay cadenas de eventos óptimas contra las cuales comparar, y la cantidad de datos es potencialmente muy grande y desorganizada. Además, al no haber tareas especificadas, se torna más difícil conocer el propósito de los usuarios. Sin embargo, existen varios beneficios sobre los métodos controlados: en primer lugar, son menos costosos dado que no requieren reclutar voluntarios ni diseñar tareas. Por otra parte, pueden hallar problemas que sólo aparecen en un ambiente real, donde los usuarios no están restringidos a realizar tareas fijas que sólo cubren parte de la funcionalidad disponible (a menos que sean diseñadas cuidadosamente para cubrir todo, pero rara vez es el caso). Por último, es importante destacar que los usuarios no están condicionados, sino que están realizando sus tareas libremente, y por lo tanto se captura más fielmente lo que sucede en realidad durante las visitas.

El trabajo de Speicher, Both y Gaedke (2015) presenta un conjunto de herramientas para obtener puntajes de usabilidad en páginas de resultados de búsqueda (SERPs – *Search Engine Results Pages*) y reparar los problemas hallados. A pesar de estar enfocado sólo en SERPs, el método también registra eventos de interacción para medir usabilidad y provee sugerencias de mejoras obtenidas de un catálogo de buenas prácticas. Una vez que una nueva versión de un SERP se crea manualmente siguiendo las sugerencias de la herramienta, se realiza un análisis comparativo entre la vieja y la nueva versión mediante un ciclo de A/B testing. Los puntajes relativos a la usabilidad se calculan según 7 factores diferentes, como comprensibilidad, distracción o densidad de información.

En un amplio estudio realizado en 2017, Vigo y Harper (2017) evalúan diferentes patrones de navegación que representan potenciales indicadores de confusión en los usuarios, basados en literatura previa y en experimentación propia. El estudio, si bien está realizado con voluntarios, busca validar estos patrones para detectar comportamiento errático sin especificar tareas. Esto permitiría que, según los resultados, tal proceso de detección pueda ser implementado luego de la validación para analizar el comportamiento de los usuarios reales, eliminando por un lado

Revista Abierta de Informática Aplicada/vol 4 (2020): 25-36
ISSN 2591-5320

la necesidad de reclutar voluntarios, y obteniendo resultados inmediatos por otro (de la misma forma de lo que se busca en el presente trabajo). Los patrones empleados buscan conductas "adaptativas", donde un usuario está esforzándose notablemente por llevar a cabo alguna tarea, ya sea porque está perdido en un camino de navegación, volviendo siempre a una misma página (empleada como lugar "seguro"), o que parece estar buscando ayuda. Una limitante para su aplicabilidad está en que la detección se realiza con un complemento de navegador (Firefox en particular), aunque podrían hallarse maneras alternativas sin restringir la instalación a la intervención del navegador. Además, no propone soluciones para los problemas hallados.

El enfoque más cercano al que aquí se propone en términos de funcionalidad es, según la investigación realizada en la bibliografía, W3Touch (Nebeling, Speicher and Norrie, 2013). Se trata de un conjunto de herramientas para evaluar problemas de interacción en interfaces táctiles de dispositivos móviles, y proveer adaptaciones para resolverlos. Incluye complejos mecanismos para registrar eventos de interacción táctil y métricas específicas para estos eventos, que por defecto son links errados (*missed links*) y nivel de zoom (*zoom levels*). W3Touch provee adaptaciones por defecto para algunos de los casos que consisten en alterar el tamaño y espaciado de los elementos afectados, dependiendo del dispositivo, y permite disparar adaptaciones creadas por desarrolladores manualmente. La principal diferencia con lo que se propone en este trabajo radica en que W3Touch está esencialmente enfocado en *responsiveness*, es decir la capacidad de las interfaces de adaptarse a diferentes dispositivos táctiles, en vez de usabilidad web en general.

3. Un proceso automatizado para usabilidad web

Para asistir a los desarrolladores en la tarea de hallar y corregir problemas de usabilidad en sus propias aplicaciones, desarrollamos un proceso alrededor de los conceptos de *usability smell* y *usability refactoring*. La idea principal fue la de maximizar el grado de automatización, requiriendo el mínimo esfuerzo posible por parte de los desarrolladores. Para lograr esto, el proceso se basa en la captura de interacción de los usuarios finales en producción como entrada principal (Grigera, Garrido, Rivero, y Rossi, 2017).

El proceso que implementa Kobold cuenta con 3 etapas. En la primera etapa se observan los eventos de interacción del lado del cliente y se capturan aquellos que resulten relevantes para el análisis de usabilidad. En la segunda etapa, los eventos de usabilidad se analizan en un servidor aparte para detectar *usability smells*. La tercera y última etapa consiste en recomendar y aplicar *usability refactorings* para resolver los *smells* detectados previamente. En algunos casos, estos *refactorings* puede aplicarse automáticamente o semiautomáticamente sumando algún dato adicional provisto por los desarrolladores. Un esquema del proceso puede verse en la Fig. 2.

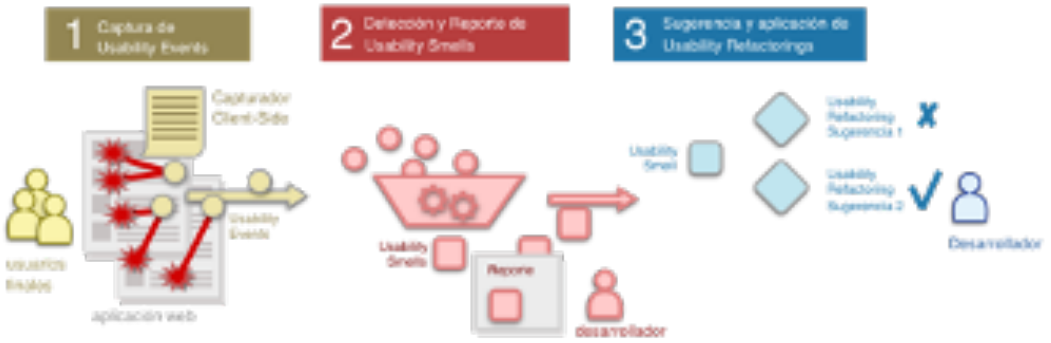

Fig. 2. Proceso general de detección de *usability smells* y aplicación de *usability refactorings*.

La tercera y última etapa es en la que los *refactorings* se sugieren y aplican. Las sugerencias de *refactorings* dependen de cada *smell* y se obtienen de catálogos existentes en los que se establecen estas relaciones (Garrido, Firmenich, *et al*., 2013; Grigera *et al*., 2016). Sin embargo, en estos catálogos, los *refactorings* suelen estar definidos de forma abstracta y difícil de llevar a código automatizado, con lo cual en este trabajo se buscó especializar estas soluciones para hacerlas más concretas, y posibilitar así la creación de *refactorings* auto-aplicables. La generación de *refactorings* es en sí una tarea compleja, pero los detalles del diagnóstico incluidos en los *smells* hacen que sea más sencilla.

3.1 Captura de Usability Events

La primera etapa consiste en capturar eventos de interacción de bajo nivel de los usuarios finales, y procesarlos para obtener *usability events.* Estos eventos son una abstracción sobre los eventos nativos JavaScript, cuyo objetivo es el de capturar información potencialmente útil para la detección de problemas de usabilidad. También sirven para no sobrecargar el servidor con eventos irrelevantes, dado que el primer filtrado y lógica de estos eventos se realiza enteramente del lado del cliente. Es importante destacar que la presencia de un *usability event* no marca la presencia de un problema de usabilidad en sí, sino que simplemente representan el material de análisis para la etapa 2. Existen más de 16 tipos de *usability events* diferentes. Un ejemplo de *usability event* es *Search*, que representa una búsqueda realizada por un usuario, capturando el momento, el término de búsqueda, el código HTML del formulario y su ubicación dentro de la página, si se hallaron o no resultados, y si alguno de los resultados fue *clickeado*, importante para luego determinar si los resultados pueden haber resultado de utilidad.

Todos los eventos de usabilidad generados por los usuarios son recolectados y analizados en el servidor, en el segundo paso del proceso. En este caso, si suficientes usuarios disparan el evento *Flash Scroll* sobre el mismo conjunto de elementos de interfaz, esto indicará la presencia del *usability smell* llamado "*Overlooked Contents*" (contenidos ignorados). Esto puede ser un signo de que cierto contenido en una página no es interesante para la mayoría de los usuarios.

Existe una relación muchos-a-muchos entre los eventos de usabilidad y los *usability smells*: un evento de usabilidad puede servir para detectar más de un *usability smell*, y en el caso opuesto, detectar un *usability smell* puede requerir del análisis de más de un tipo de evento de usabilidad.

3.2 Detección de Usability Smells

La segunda etapa del proceso es el momento en que los *usability events* se procesan para capturar *usability smells*. Para cada tipo de *smell*, existe un componente individual que consume y analiza uno o más tipos de eventos. Por ejemplo, para detectar el *smell Scarce Search* Results (resultados de búsqueda escasos), existe un componente que analiza los eventos *Search* y calcula la proporción de búsquedas con y sin resultados. Los algoritmos para detectar cada *smell* están implementados en entidades llamadas *usability smell finders* (buscadores de *usability smells*), y cada uno de estos buscadores es capaz de detectar un único tipo de *smell*, consumiendo y analizando uno o más tipos de eventos de usabilidad. Cada buscador está configurado con ciertos parámetros que determinan el número, proporción, o combinación de eventos de usabilidad que desencadenan la presencia de un *smell* particular. Los valores por defecto fueron hallados mediante experimentación con la herramienta, pero se pueden ajustar para servir propósitos específicos.

El conjunto de buscadores actualmente implementados puede extenderse fácilmente. La arquitectura del sistema está preparada de tal manera que agregar un buscador para una nueva clase de *usability smell* sólo implica seleccionar el tipo o los tipos de *usability event* que deben ser consumidos, el criterio para agruparlos (se pueden elegir varias estrategias implementadas o implementar una nueva, por defecto es el mismo elemento DOM), y la lógica de detección, que típicamente involucra contar proporciones sobre ciertas propiedades de los eventos.

3.3 Aplicación de Refactorings *y Versionamiento*

A partir de los problemas hallados en la etapa de detección de *usability smells*, es posible sugerir soluciones en forma de *refactorings* de usabilidad, para que los desarrolladores puedan ir aplicando soluciones incrementalmente a los problemas reportados. Gracias a los catálogos preexistentes que sirvieron como punto de partida, conectar los *usability smells* con las soluciones (*usability refactorings*), no es una tarea compleja.

La información requerida para generar automáticamente el código de los *usability refactorings* surge de los mismos *usability smells* que intentan solucionar. Por ejemplo, el *smell Free Input for Limited Values* explicado en la sección anterior puede solucionarse aplicando un A*dd Autocomplete*, que es un *refactoring* que agrega a un campo de texto libre la capacidad de sugerir términos para autocompletado. En este caso, los valores que se sugieren en el autocompletado se obtienen del listado de valores frecuentes que ingresaron los usuarios antes de detectar el *usability smell*.

Es importante recordar que para un mismo *smell* puede haber múltiples sugerencias de *refactorings*. Por ejemplo, consideremos que una interfaz incluye un campo de texto libre para ingresar una fecha, lo que se implicaría el *smell Unformatted Input*, que señala que los usuarios tienen que encargarse de aplicar manualmente el formato a la fecha. Este problema puede resolverse al menos de tres maneras: incorporando un calendario (*refactoring Add Datepicker*), agregando 3 listas de selección para día, mes y año (*refactoring Turn Input into Selects*), o bien incorporando formato automático de tipo "máscara" (*Format Input*). Dependiendo del tipo de fecha a ingresar, puede resultar mejor uno que otro. Una fecha de cumpleaños, por ejemplo, puede ser más fácil ingresar textualmente o mediante listas de selección o incluso con un campo de texto con asistencia de formato, ya que los usuarios generalmente la conocen de memoria. Para una fecha futura, como por ejemplo para una reserva de un hotel, podría ser más conveniente el calendario, ya que da un mejor contexto al usuario, y muestra los días de la semana.

Dado que los desarrolladores tendrán a veces que elegir entre varias sugerencias de refactoring, o que necesitan probar los refactorings antes de utilizarlos, la aplicación permite crear versiones de la interfaz de la aplicación bajo análisis, y elegir diferentes refactorings para cada versión (Grigera, J., Gardey, J. C. *et al.,* 2018). Esto permite crear ambientes de prueba, con el objetivo de validar si una solución elegida sirve como tal, o para comparar diferentes alternativas.

4. Kobold

Empleando las ideas y arquitectura presentadas anteriormente, desarrollamos Kobold, un servicio para hallar y corregir *smell*s de usabilidad en aplicaciones puestas en producción. Para utilizarlo, los desarrolladores se registran con una cuenta y obtienen un pequeño bloque de código JavaScript. Este código habilita la captura de eventos de interacción que luego se analizarán para detectar *usability smells* en tiempo real. Una vez instalado, el desarrollador podrá ver los *smells* capturados en su cuenta. Desde allí, también puede ver las sugerencias de *refactorings*, y puede seleccionar aplicar uno de ellos. Una vez seleccionado el *refactoring*, Kobold lo aplica en el sitio bajo análisis, en producción. Esto es posible gracias al mismo código que habilita la captura de eventos: el componente *client-side*. Este componente se comunica con el servidor para obtener los refactorings y reescribe el código HTML/CSS de la aplicación web al momento de la carga.

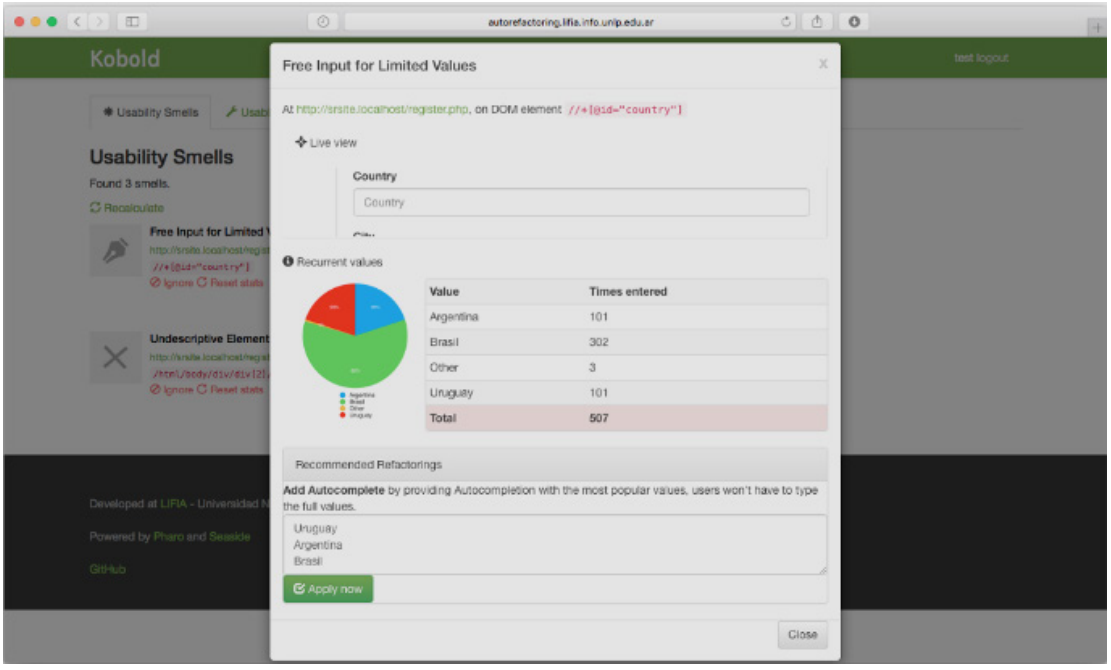

Fig. 3. Captura de Kobold presentando un reporte de *usablity smell* junto al *refactoring* asociado.

La Fig. 3 muestra una captura de Kobold sugiriendo el refactoring *Add Autocomplete* (agregar autocompletado) para corregir el *usability smell* "*Free Input for Limited Values*". La captura muestra el diagnóstico del *smell* con una versión en vivo del *element* afectado (el campo de texto "*Country*") y pide confirmación al usuario sobre la aplicación del *refactoring*, permitiendo editar los valores del autocompletado.

Revista Abierta de Informática Aplicada/vol 4 (2020): 25-36
ISSN 2591-5320

Para la aplicación de *refactorings*, Kobold permite generar diferentes versiones de la misma aplicación en producción, cada una con su propia URL privada (que es en realidad una variante de la URL de producción). Esto tiene la ventaja de poder realizar pruebas sin modificar la aplicación pública, pero explorar alternativas de forma privada utilizando la misma infraestructura de producción, minimizando al extremo la necesidad de recursos externos.

5. Validaciones

En esta sección se detallan las validaciones realizadas para comprobar la correcta captura de *usability events*, la detección de *usability smells* y finalmente la aplicación automatizada de *usability refactorings*.

5.1. Validación de Usability Events

La captura de eventos de interacción del lado del cliente ("*usability events*") es la base de la detección de *usability smells*. Algunos de ellos involucran para su detección heurísticas que pueden ser proclives al error, como interpretar la intención del usuario (*"¿quiso realmente el usuario hacer click en este elemento?"*) o inferir el significado de un elemento de interfaz ("*esto parece ser un formulario de búsqueda*"). Para analizar si los eventos capturados reflejan realmente lo que sucede durante la interacción, se realizaron dos validaciones separadas: por un lado, las heurísticas que buscan interpretar las intenciones del usuario (grupo 1), y por el otro, las que tienen que ver con inferir el propósito de un elemento DOM de interfaz (grupo 2). Otras heurísticas son suficientemente simples o no involucran ninguna interpretación subjetiva, con lo cual no fue necesario incluirlas en el experimento (Grigera *et al.*, 2017).

1. Heurísticas que involucran intención del usuario: para validar las heurísticas del grupo 1, se pidió a un grupo de voluntarios que llevaran a cabo interacciones específicas sobre diferentes aplicaciones web, y luego se compararon los eventos de usabilidad capturados por la herramienta con la observación directa, "manual" de dichas interacciones.
2. Heurísticas que involucran propósito de elementos DOM: para la validación del grupo 2, es decir, las heurísticas basadas en la interpretación de elementos DOM, no fue necesario reclutar voluntarios, dado que su interpretación no está en juego. Estas heurísticas se validaron a través de diferentes sitios web para comprobar la capacidad de la herramienta de detectar la misma situación en contextos diferentes.

Para analizar los resultados se empleó un *score* F2 que consolida métricas de precisión y cobertura (*precision and recall*) del funcionamiento de la herramienta. Se obtuvieron valores F2 altos para todas las heurísticas de captura que dependen de la interpretación automatizada, en promedio **0.7925** (desvío estándar **0.09392**). El grupo que mejores puntajes obtuvo (0.75 < F2 <0.97 para *Click Attempt*, *Tooltip Attempt*, *Flash Scroll*, *Navigation Path*, *Bulk Action*) coincide mayormente con los eventos de usabilidad que dependen de la interpretación del usuario. El grupo con resultados más moderados (.67 < F2 < .75, *Search*, *Flash Navigation*, *Form Submission*) se corresponde mayormente con los eventos que involucran razonamiento sobre estructuras de elementos DOM.

5.2. Validación de captura de Usability Smells

Para evaluar la detección automatizada de *usability smells*, se analizó el comportamiento de la herramienta en sitios reales durante la interacción de voluntarios, ya sea de forma independiente, o en comparación con hallazgos realizados por estudios de usuario con expertos en un ambiente controlado (Grigera *et al.*, 2017). El método empleado para este experimento fue Goal–Question–Metric (GQM) (Basili, Caldiera and Rombach, 1994), que propone la definición de metas (*goals*), refinarlas en preguntas (*questions*) y especificar las métricas (*metrics*) requeridas para responder estas preguntas.

Los voluntarios tuvieron que completar 5 tareas típicas en 2 aplicaciones reales de *e-commerce*, mientras se observó su comportamiento y se midieron los tiempos de completado. Luego de los tests, se obtuvo una lista de problemas de usabilidad para ambas aplicaciones y también los tiempos y recursos empleados por los desarrolladores y el personal. La segunda parte del experimento consistió en recolectar resultados de Kobold. Para obtenerlos, se creó una cuenta para cada sitio, esta vez la versión real de producción, y se instaló el script tal como lo indican las instrucciones de la herramienta.

Respecto a los resultados, la herramienta fue capaz de hallar **66.67%** del total de problemas encontrados, mientras que mediante el test manual se halló el **70.37%**, con una intersección del **37.04%**. Respecto a esta intersección, si sólo consideramos problemas hallados manualmente, Kobold encontró **52.63%**.

5.3. Validación de Usability Refactorings

Durante la validación de *Usability Refactorings* se comparó la usabilidad en uso de aplicaciones refactorizadas mediante la herramienta contra sus versiones originales, a través de tests de usuario (Grigera, J., Garrido, A., & Rossi, G. 2017).

Durante este nuevo experimento se aplicaron *refactorings* para resolver los *usability smells* hallados en el experimento previo. Los *refactorings* se aplicaron mediante la herramienta Kobold. En particular se aplicaron instancias de *Add Autocomplete*, *Add Validation*, *Turn Attribute into Link*, *Add Default Value* y *Add Processing Page*. La idea del experimento fue la de comparar los niveles de usabilidad sobre las aplicaciones antes y después de refactorizar.

Se midieron tres aspectos relacionados a la usabilidad de acuerdo con el estándar ISO/IEC 25010: efectividad en términos de porcentaje de completado, eficiencia en tiempo promedio de completado, y satisfacción por medio de un cuestionario SUS estándar.

Se concluyó que las versiones refactorizadas siempre mostraron mejoras en los tres aspectos evaluados: efectividad (8.33% de mejora en promedio), eficiencia (18% menos tiempo en promedio) y satisfacción (6.88%).

6. Conclusiones

En este artículo se presentó Kobold, un servicio para aplicar mejoras incrementales de usabilidad web de manera automatizada, con la intención de minimizar tanto los recursos necesarios para

aplicarlo, como el esfuerzo de instalación. Se mostró que es posible, en algunos casos, ofrecer soluciones automatizadas para los problemas de usabilidad detectados utilizando la técnica de *refactoring* del lado del cliente (CSWR).

Si bien el rango de problemas que puede hallar está limitado a lo que puede automatizarse (es decir, sin necesidad de interpretación subjetiva), la metodología ofrece múltiples ventajas: en primer lugar, el esfuerzo de instalación es insignificante. En la herramienta implementada, sólo es necesario completar un formulario de registro y pegar un bloque de código en la aplicación bajo análisis. Esto baja la barrera de la adopción al mínimo posible. Se presentó además una validación que muestra, por un lado, el grado de precisión en la captura de problemas en términos de *usability smells*, y por el otro el grado de mejora que se puede obtener a partir de la aplicación de CSWRs.

Actualmente nos encontramos trabajando en dos nuevas líneas: por un lado, estamos buscando una manera de medir el nivel de éxito de los *refactorings* mediante una técnica de *Machine Learning*. Por otro lado, estamos trasladando la metodología a la interacción en interfaces móviles, que representan hoy la mayoría del tráfico global de internet.

Bibliografía

» Apaolaza, A., Harper, S. and Jay, C. (2015). 'Longitudinal Analysis of Low-Level Web Interaction through Micro Behaviours', in *Proceedings of the 26th ACM Conference on Hypertext & Social Media - HT '15*. New York, New York, USA: ACM Press, pp. 337–340. doi: 10.1145/2700171.2804453.

» Apaolaza, A. and Vigo, M. (2017). 'WevQuery', *Proceedings of the ACM on Human-Computer Interaction*. ACM, 1(1), pp. 1–17. doi: 10.1145/3095806.

» Chi, E. H. (2002). 'Improving Web Usability Through Visualization', *IEEE Internet Computing*, 6(2), pp. 64–71. doi: 10.1109/4236.991445.

» Garrido, A., Rossi, G. and Distante, D. (2011). 'Refactoring for Usability in Web Applications', *IEEE Software*, 28(3), pp. 60–67. doi: 10.1109/MS.2010.114.

» Garrido, A., Firmenich, S., *et al.* (2013). 'Personalized web accessibility using client-side refactoring', *IEEE Internet Computing*, 17(4), pp. 58–66.

» Grigera, J., Garrido, A., Panach, J. I., Distante, D., & Rossi, G. (2016). Assessing refactorings for usability in e-commerce applications. Empirical Software Engineering, 21(3), 1224-1271.

» Grigera, J., Garrido, A., Rivero, J. M., & Rossi, G. (2017). Automatic detection of *usability smells* in web applications. International Journal of Human-Computer Studies, 97, 129-148.

» Grigera, J., Garrido, A., & Rossi, G. (2017). Kobold: web usability as a service. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 990-995). IEEE.

» Grigera, J. (2018). *Self-Refactoring: mejoras automáticas de usabilidad para aplicaciones web* (Doctoral dissertation, Universidad Nacional de La Plata).

» Grigera, J., Gardey, J. C., Garrido, A., & Rossi, G. (2018b). Live versioning of web applications through refactoring. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (pp. 872-875).

» Nebeling, M., Speicher, M. and Norrie, M. (2013b). 'W3touch: Metrics-based Web Page Adaptation for Touch', *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI '13*, p. 2311. doi: 10.1145/2470654.2481319.

» Rubin, J. & Chisnell, D. (2008). *Handbook of Usability Testing: Howto Plan, Design, and Conduct Effective Tests*. Wiley.

» Saadawi, G. M. *et al.* (2005). 'A Method for Automated Detection of Usability Problems from Client User Interface Events AMIA 2005 Symposium Proceedings Page - 654 AMIA 2005 Symposium Proceedings Page - 655', pp. 654–658.

» Speicher, M., Both, A. and Gaedke, M. (2015). 'S.O.S.: Does Your *Search* Engine Results Page (SERP) Need Help?', in *Proc. ACM Conf. on Human Factors in Comp Systems - CHI '15*. New York: ACM Press, pp. 1005–1014. doi: 10.1145/2702123.2702568.