Métodos y herramientas de desarrollo y evaluación de refactorings para la mejora de la experiencia de usuario en aplicaciones web

Juan Cruz Gardey

Facultad de Informática
Universidad Nacional de La Plata

Directora: Dra. Alejandra Garrido
Co-Director: Dr. Julián Grigera
Co-Director: Dr. Gustavo Rossi

Tesis realizada para obtener el grado de
Doctor en Ciencias Informáticas

Resumen

La experiencia de usuario (UX) es un factor crucial que determina el éxito de las aplicaciones web. Si bien actualmente se reconoce la importancia de proveer una experiencia de usuario óptima, las prácticas de evaluación y mejora de la UX suelen postergarse. Más allá de los recursos necesarios como expertos en UX y usuarios para hacer pruebas, el tiempo requerido por estas prácticas es el principal motivo por el cual resulta difícil integrarlas en los métodos ágiles ampliamente utilizados hoy en día.

Para facilitar la evaluación de la UX, se han desarrollado métodos automáticos para detectar y solucionar problemas de interacción. Uno de los métodos propuestos es la técnica de refactoring de usabilidad, la cual se basa en aplicar pequeñas transformaciones (refactorings) a través scripts ejecutados en el navegador que sin alterar la funcionalidad de la aplicación pretenden mejorar la UX. Sin embargo, esta técnica fue propuesta como una solución automática, en la que los refactorings aparecen como soluciones definitivas sin antes poder validar cómo impactan en los usuarios.

En esta tesis se desarrolla un conjunto de métodos y herramientas para dar soporte a los expertos en UX en las tareas de evaluación y mejora de la experiencia de usuario. En primer lugar se presenta un método para explorar alternativas de diseño haciendo uso de la técnica de refactoring. Este método se desarrolla en una herramienta llamada UX-Painter, mediante la cual un diseñador puede crear distintas versiones de una aplicación web utilizando los refactorings para generar los cambios de diseño, sin la necesidad de codificarlos. De esta manera, es posible visualizar, validar y evaluar cambios de diseño en la misma aplicación web sin que éstos tengan que ser implementados.

Respecto a la evaluación de los diseños, no solo es importante determinar el impacto de un cambio de diseño en la experiencia del usuario, sino que además resulta fundamental poder comparar distintas alternativas. Sobre todo porque los refactorings en ciertos casos ofrecen más de una solución posible para un mismo problema. Por este motivo, en esta tesis también se desarrolla una métrica denominada *esfuerzo de interacción* que permite evaluar y comparar diseños alternativos en páginas web. Se trata de un puntaje asignado por expertos en UX sobre los distintos widgets de una página web, que son aquellos elementos que se ven modificados por los refactorings. Para calcularlo automáticamente, se desarrollaron modelos de predicción que toman como entrada diferentes logs de interacción de usuario que reciben el nombre de micro-medidas. El hecho que la métrica sea transparente a los usuarios hace que pueda ser utilizada para evaluar diseños con múltiples usuarios en experimentos online

como A/B testing o similares. Como prueba de concepto, se implementa UX-Analyzer, una herramienta web que permite calcular y visualizar el esfuerzo de interacción de distintas versiones de una aplicación web.

Finalmente, una vez que se verifica que ciertos refactorings mejoran la UX, deben ser implementados en la aplicación bajo análisis. Para facilitar esta tarea a los desarrolladores, se presenta un mecanismo para generar una implementación preliminar de los refactorings usando como base los frameworks y librerías utilizados actualmente para el desarrollo de interfaces web.

El trabajo incluye distintas validaciones que comprueban la aplicabilidad de los desarrollos propuestos para facilitar la mejora de la UX.

Agradecimientos

En primer lugar quiero agradecerle a Alejandra por acercarme la propuesta para hacer un doctorado y por ser mi guía durante todo el trabajo.

También quiero agraderle a mis codirectores Julián y Gustavo. A Juli por su colaboración constante, y a Gustavo por estar ahí y darme un consejo cada vez que lo necesitaba.

A Andrés Rodríguez que tuvo muchísimo que ver con esta investigación.

A toda la gente del LIFIA que desde el primer día me abrieron las puertas y me dieron muchas oportunidades para crecer.

Finalmente a mi familia por el apoyo todo este tiempo. En especial a Virginia que me banca todos los días.

Índice general

1. Introducción **9**

1.1. Motivación 9
1.2. Objetivos 13
1.3. Contribución 14
1.4. Organización del documento 15

2. Marco teórico y trabajo relacionado **17**

2.1. Experiencia de Usuario 17
2.2. Métodos ágiles 19
2.2.1. Refactoring de código y deuda técnica 19
2.2.2. Integración de métodos ágiles con métodos de evaluación de UX . . 20
2.3. Refactorings de UI 21
2.4. Adaptación web 23
2.5. End-user development 24
2.6. Herramientas para explorar cambios de diseño 25
2.7. Métodos de evaluación de UX 26
2.8. Interactividad y carga cognitiva 29
2.9. Aprendizaje automático 30
2.9.1. Árboles de decisión 31
2.9.2. Aprendizaje automático en UX 33

3. Un método para desarrollar y evaluar UX refactorings **35**

3.1. Solución propuesta 35
3.1.1. Generación de variantes de diseño 37
3.1.2. Evaluación y comparación de variantes de diseño 38
3.2. Proceso de mejora de la experiencia de usuario 39

4. Variantes de diseño de UI a través de refactorings **42**
4.1. Refactorings para explorar variantes de diseño . . . 42
4.2. UX-Painter . . . 43
4.2.1. Contexto . . . 43
4.2.2. Requerimientos . . . 44
4.2.3. Catálogo de refactorings . . . 46
4.2.4. Adaptación de estilos CSS . . . 51
4.2.5. Arquitectura . . . 54
4.2.6. Funcionamiento . . . 55
4.3. Generación de código de refactorings . . . 61
4.4. UX-Painter en un ciclo de desarrollo ágil . . . 64

5. Métrica para medir el impacto de los UX refactorings **66**
5.1. Evaluación de refactorings . . . 67
5.2. Esfuerzo de interacción . . . 68
5.3. Micro-medidas . . . 71
5.3.1. Proceso de selección . . . 71
5.3.2. Widgets . . . 72
5.4. Modelos de predicción . . . 77
5.4.1. Recolección de datos . . . 77
5.4.2. Implementación . . . 82
5.4.3. Importancia de las micro-medidas . . . 84
5.5. Discusión . . . 90

6. Visualización del esfuerzo de interacción **93**
6.1. Esfuerzo de interacción global . . . 93
6.2. UX-Analyzer . . . 95
6.2.1. Evaluaciones . . . 95
6.2.2. Versiones . . . 96
6.2.3. Sesiones . . . 98
6.2.4. Widgets . . . 99
6.3. Discusión . . . 100

7. Evaluación **103**
7.1. UX-Painter . . . 103
7.1.1. Evaluación del método . . . 103

7.1.2. Evaluación de usabilidad 108
7.1.3. Discusión 109
7.2. Esfuerzo de interacción 110
7.2.1. Métricas utilizadas 111
7.2.2. Recolección de datos 112
7.2.3. Resultados 114
7.2.4. Discusión 117

8. Conclusiones y trabajo futuro **119**
8.1. Contribuciones 120
8.2. Trabajos futuros 123
8.3. Publicaciones científicas 125

Índice de figuras

2.1. Ejemplo de árbol de decisión (muy simple) para clasificar animales. 32

3.1. Refactorings alternativos para el usability smell *Unformatted Input*. 36
3.2. Proceso de mejora de la UX asistido por las herramientas desarrolladas. . . 41

4.1. El refactoring Date Input into Selects en el campo "ida"produce una alteración no deseada en el layout de la página. 52
4.2. El estilo del nuevo menú desplegable "Ciudad"no coincide con el del menú desplegable de "País". 53
4.3. Opciones de estilo para *Nuevo Link*. La primer opción es la del link más próximo, *Empretec*. Los estilos de *Arrepentimiento Baja* y *Sucursales y Cajeros* son la segunda y tercera opción, respectivamente. 54
4.4. Opciones de estilo para el nuevo Datepicker. La primer opción que es la que se ve en el título y en los días del Datepicker. Ésta tiene peso 7 porque los 7 links de la barra de navegación utilizan estos colores. Por otro lado, la segunda opción posee un peso de 2 porque hay 2 elementos que tienen esa combinación de colores. 55
4.5. Listado de versiones de la aplicación actual. La primera vez que se abre la herramienta solo se ve la versión original que no puede modificarse. 56
4.6. Listado completo de CSWRs. Cada refactoring tiene una opción de más información para que el usuario pueda entender mejor qué modificaciones realiza. 56
4.7. Descripción de Add Autocomplete. Se provee una breve descripción junto con un ejemplo del refactoring 57
4.8. Selección del elemento a refactorizar. Solo se pueden seleccionar los elementos refactorizables con el CSWR elegido. 58
4.9. Parámetros adicionales de Add Inline Form Validation. El usuario debe seleccionar los campos requeridos. 58

4.10. Vista previa de Add Inline Form Validation. 59
4.11. Vista previa de Turn Select into Autocomplete. Las opciones de estilo permiten cambiar el color de los valores sugeridos utilizando los colores que más se repiten en la página. 60
4.12. Versión alternativa con los todos los refactorings aplicados. 60
4.13. En cada versión se agrega una opción para ver una implementación de la misma. 62
4.14. Vista del código generado para un versión en la que se listan los componentes de ReactJS. 63
4.15. UX-Painter en un contexto de desarrollo ágil. 65

5.1. Esfuerzo de interacción. El experto en UX observa la interacción del usuario con un widget y decide el puntaje de la misma. 69
5.2. Modelos de predicción de los widgets campo de texto y menús desplegables. 70
5.3. Ejemplo del área de un widget. Este área se calcula agregándole al rectángulo ocupado por el widget, un margen en cada uno de los lados (izquierda, derecha, arriba y abajo). 74
5.4. Extensión web que instalaron los participantes para capturar las interacciones. 80
5.5. Aplicación web utilizada por los expertos para la asignación de puntajes. A la izquierda se muestra la grabación de pantalla y a la derecha el listado de widgets cuya interacción debe ser puntuada. 81

6.1. Vista principal de UX-Analyzer que muestra las evaluaciones del usuario. . 96
6.2. Vista de una versión recién creada. Allí se muestra el script que el usuario debe insertar en su aplicación para capturar las interacciones. 97
6.3. Lista de versiones correspondientes a una evaluación. 97
6.4. Esfuerzo de interacción de una versión que se obtiene promediando los valores de cada interacción de usuario con cada widget. 98
6.5. Detalles de una versión que contiene sesiones de usuario capturadas. 99
6.6. Listado de sesiones de usuario capturadas en una versión. 99
6.7. Listado de widgets incluidos en las sesiones de usuario capturadas. 100

7.1. Frecuencia de modificación de una UI para mejorar la UX. 104
7.2. Medios utilizados para evaluar los diseños. 106
7.3. Imagen obtenida de [Sauro and Kindlund, 2005b] que resume cómo se obtiene el puntaje de SUM. 112

7.4. Imagen obtenida de [Katsanos et al., 2013] que muestra KLM-Form Analyzer. 113
7.5. Ejemplo de una página en la que se calculó el esfuerzo de interacción, junto con las opciones de *Iniciar* y *Abandonar* la tarea. 114
7.6. Gráfico que compara los resultados de esfuerzo de interacción (normalizados) con los de SUM. Adicionalmente, se muestran los niveles de satisfacción. . . 117

Índice de tablas

5.1. Cantidad de interacciones recolectadas para cada tipo de widget. 82
5.2. Resultados de los modelos de cada tipo de widget. 84
5.3. Gini importance promedio de las micro-medidas de los campos de texto. . . 86
5.4. Gini importance promedio de las micro-medidas de los menús desplegables. 87
5.5. Gini importance promedio de las micro-medidas de los links. 88
5.6. Gini importance promedio de las micro-medidas de los botones de radio. . . 88
5.7. Gini importance promedio de las micro-medidas de los datepicker. 89
5.8. Gini importance promedio de los menús desplegables de fecha. 90

7.1. Esfuerzo de interacción de las 5 páginas web. 115
7.2. SUM de cada página web, que resulta de promediar los 4 valores normalizados. 116
7.3. Resultados de KLM junto con la cantidad de widgets de cada formulario y el tiempo normalizado por esta cantidad. 116

Capítulo 1

Introducción

Este capítulo introduce los fundamentos y la propuesta de este trabajo. Primero, se explica la motivación y la problemática que se pretende resolver. Luego, se establecen los objetivos concretos y se define la contribución. Finalmente, se resume y se explica la estructura de este documento.

1.1. Motivación

Actualmente las aplicaciones web son ampliamente utilizadas para la realización de distintas actividades como comprar productos, interactuar socialmente, leer noticias o planificar viajes. Si bien el avance de la tecnología web permite desarrollar soluciones cada vez más complejas y atractivas para los usuarios, lo cierto es que muchas veces estas aplicaciones proveen una experiencia de usuario pobre. El término experiencia de usuario (UX) se refiere a las percepciones que los usuarios tienen acerca de un producto o servicio que no sólo dependen de factores relativos al diseño de éste, sino que además se ven influenciadas por aspectos subjetivos de los usuarios como sus emociones y sentimientos. Algunos ejemplos en la web de cuestiones que contribuyen a una experiencia de usuario pobre son: páginas sobrecargadas de contenido, información mal distribuida, procesos de negocio inconsistentes, problemas para enviar formularios, diseños poco elegantes, entre muchos otros.

Las empresas reconocen el papel fundamental que juega la UX en el éxito de una aplicación o producto, pero con frecuencia su evaluación es dejada de lado debido al alto costo que esto implica. La evaluación de la UX no solamente suele requerir diferentes recursos como por ejemplo expertos para descubrir problemas que afectan a la UX y usuarios finales para realizar pruebas, sino que además es una actividad que consume mucho tiempo, algo

que no abunda en los procesos de desarrollo utilizados hoy en día.

Desde hace un tiempo, los métodos ágiles son el medio más usado para el desarrollo de software [Beck, 2000], porque su objetivo principal es atender rápidamente las necesidades de los clientes. Al estar enfocados en entregar software lo antes posible, los métodos ágiles no llegan a darle a la UX la importancia que merece. Esta falta de atención a la UX, hizo que el término Diseño Centrado en el Usuario (DCU) cobrara una gran relevancia en los últimos años. DCU se trata de una filosofía de diseño que pone a los usuarios finales en el centro de la escena a través del estudio minucioso de sus necesidades y del desarrollo de productos para satisfacerlas, garantizando una experiencia óptima. A partir de las diferencias de los métodos ágiles y DCU, ha surgido la necesidad de integrar ambos enfoques con el objetivo de no perder la dinámica de los métodos ágiles pero al mismo tiempo incluir al usuario final en el desarrollo. El problema para lograr esta integración radica en que los dos métodos manejan tiempos muy diferentes, lo cual hace que sea difícil sincronizar sus prácticas. Por ejemplo, DCU reserva un tiempo considerable al análisis y diseño antes de que comience el desarrollo, produciendo mucha documentación, al contrario de los métodos ágiles que procuran mantener la documentación mínima y necesaria.

Esta dificultad para introducir la evaluación de la UX en los métodos ágiles pone en evidencia la necesidad de desarrollar métodos para que los desarrolladores y diseñadores de UX puedan trabajar en forma conjunta. En este sentido, uno de los aspectos en los que se viene trabajando es en automatizar la evaluación de la UX para reducir sus costos, lo cual implica no solamente ayudar en la detección de problemas, sino también en dar soluciones a éstos. Con respecto a la detección automática de problemas, uno de los métodos que se ha propuesto es el análisis de los eventos de interacción que generan los mismos usuarios [Ivory and Hearst, 2001]. A partir de este análisis, es posible identificar patrones que contribuyen a una interacción deficiente. Un ejemplo de este tipo de análisis se presenta en el trabajo de Atterer et al., en el que se captura la actividad del mouse para identificar patrones de uso en las aplicaciones web [Atterer et al., 2006]. Sin embargo, la principal limitación de este tipo de propuestas es que se necesitan expertos en UX para analizar los datos recolectados y para elaborar soluciones a los problemas que se identifiquen.

En respuesta a la limitación anterior, la técnica de refactoring de usabilidad (usability refactoring) [Garrido et al., 2011] fue propuesta como un medio para aplicar mejoras de usabilidad incrementalmente mediante transformaciones en la interfaz de usuario (UI) cuyo objetivo es resolver problemas catalogados como malos olores de usabilidad (usability smells). Siguiendo la filosofía del refactoring de código, cada refactoring de usabilidad es una transformación muy concreta de la UI que apunta a mejorar la interacción del usuario

sin alterar la funcionalidad de la aplicación subyacente. El hecho de contar con soluciones predefinidas para un conjunto de problemas de usabilidad, dio lugar al desarrollo de herramientas para detectar automáticamente los usability smells y asistir en la aplicación de refactorings como solución. En particular, la herramienta Kobold reporta los usability smells de una aplicación web a partir del análisis de eventos de interacción los usuarios, que luego pueden solucionarse a través de scripts ejecutados en el navegador del usuario denominados Client-side web refactorings (CSWRs) [Grigera et al., 2017b].

Los refactorings de usabilidad y en particular los CSWRs dan la posibilidad de introducir mejoras en la experiencia de usuario a muy bajo costo. De hecho una herramienta como Kobold que tiene la habilidad de sugerir y aplicar modificaciones, es especialmente atractiva para equipos de desarrollo que no cuentan con expertos en UX. No obstante, se debe tener en cuenta que los CSWRs se aplican directamente en la UI que se encuentra en producción, por lo cual los cambios que se introducen son visibles a todos los usuarios. Y aunque los CSWRs en general mejoran la UX, puede ocurrir que en algunos casos produzcan un efecto adverso, ya sea por ejemplo porque los usuarios están acostumbrados a un diseño particular o porque la ejecución de un CSWR generó una modificación no deseada en la UI. Otra cuestión a considerar es que hay usability smells que pueden solucionarse con CSWRs alternativos y en esos casos el usuario debe elegir a su juicio qué alternativa utilizar, por lo cual puede que la opción elegida no sea la más apropiada para un contexto particular.

Esta cuestión de no poder determinar a priori el impacto de los CSWRs en los usuarios, sirve como justificación a la necesidad de visualizar y evaluar de algún modo los cambios de diseño antes de aplicarlos definitivamente. Por este motivo, parte de lo que se propone en esta tesis se basa en utilizar los CSWRs para generar variantes de diseño que puedan ser evaluadas y así finalmente poder implementar cambios de diseño que efectivamente mejoren la UX. La exploración de alternativas de diseño es una práctica común en el flujo de trabajo de los diseñadores de UX, que en general se realiza sobre artefactos como mockups o prototipos [Garcia et al., 2019]. El hecho que se utilicen estos artefactos en parte tiene que ver con que muchas veces resulta imposible explorar las variantes en la UI real porque se requiere implementarlas y no se cuenta con la asistencia de desarrolladores para esta tarea. Sin embargo, aunque los prototipos son muy útiles para visualizar y validar ideas en la etapa de diseño inicial de una UI, es probable que en ellos no se reflejen ciertos aspectos de la interacción de los usuarios. Dada la importancia de evaluar los diseños en el contexto de uso real, en este trabajo se propone generar las variantes de diseño en el navegador, lo que da la posibilidad de realizar de pruebas con usuarios ya que la funcionalidad de la aplicación no se ve modificada. Para poder realizar cambios de diseño se utilizan los CSWRs, que se

aplican de manera asistida para generar los cambios rápidamente sin tener que codificarlos.

Con respecto a la evaluación automática de las interfaces de usuario, la mayoría de los métodos propuestos hasta el momento tienen como objetivo evaluar una UI sin necesitar de usuarios que interactúen con ésta. Estos métodos incluyen modelos de simulación como KLM-GOMS para estimar el tiempo requerido por una tarea [Card et al., 1980], y modelos de aprendizaje automático que sirven para predecir distintos aspectos de la UX como la performance del usuario [Li et al., 2018] y otras propiedades como complejidad visual, percepción de colores, entre otros [Oulasvirta et al., 2018]. Si bien el hecho de no tener que reclutar usuarios reduce considerablemente el costo de la evaluación, hay cuestiones que deben evaluarse analizando la interacción del usuario con la UI. Por otro lado, las herramientas que sí analizan la interacción como el caso de aquellas que permiten realizar experimentos de A/B testing y similares para comparar alternativas, tienen la particularidad que utilizan métricas que están más enfocadas en medir el desempeño de un negocio que en la evaluación de la UX [Nielsen, 2005]. Al respecto de esto, Speicher et al. han propuesto utilizar un mecanismo similar al de A/B testing pero para comparar la usabilidad de páginas de resultados de búsqueda [Speicher et al., 2015]. La herramienta que han desarrollado realiza una captura automática de interacciones del usuario y las analiza en base a heurísticas de usabilidad predefinidas para lograr una puntuación global de las páginas analizadas. La mayor limitación de este enfoque, además que solo se aplica a páginas de resultados de búsqueda, es que las heurísticas utilizadas resultaron fuertemente dependientes de variables del contexto del usuario, como la tarea que está realizando o incluso la resolución y el tamaño de la pantalla. Para evaluar la experiencia del usuario en aplicaciones web y en particular el impacto que tienen los CSWRs en una UI, en esta tesis se desarrolla una métrica llamada *esfuerzo de interacción*. Esta métrica corresponde a un puntaje asignado por expertos en UX que representa el esfuerzo requerido por los usuarios para interactuar con los distintos widgets de una UI. Debido a que es un puntaje subjetivo, para poder calcularlo automáticamente se desarrollaron modelos de aprendizaje automático que permiten predecirlo utilizando logs de interacción que reciben el nombre de *micro-medidas*. La ventaja del esfuerzo de interacción es que al ser un puntaje que se calcula sobre los widgets individuales de una UI, no solo permite localizar los problemas de interacción sino que además facilita la comparación de widgets alternativos que sirven para una misma tarea. Además, el cálculo de la métrica es completamente transparente a los usuarios, lo cual da la posibilidad de hacer pruebas con múltiples usuarios como en los experimentos A/B testing, con la particularidad que se evalúa UX y no las ganancias de un negocio.

Como se indicó anteriormente, la idea de generar rápidamente variantes de diseño y

evaluarlas utilizando el esfuerzo de interacción, responde a la necesidad de comprobar que ciertos cambios de diseño mejoren la experiencia de usuario sin que éstos tengan que ser implementados. Esto se debe a que la implementación de modificaciones suele requerir mucho tiempo, algo que no abunda dentro de los ciclos de desarrollo ágil en los que los desarrolladores están centrados principalmente en la incorporación de nuevas funcionalidades. Sin embargo, aunque la codificación sea costosa, una vez que se verifica que los cambios tienen un efecto positivo deben ser implementados en la aplicación bajo análisis. Esto quiere decir que las modificaciones impuestas por los CSWRs utilizados para generar variantes de diseño en algún momento tendrán que ser manualmente codificadas e integradas en el código fuente de la aplicación. Para facilitar este proceso de codificación, en este trabajo también se realiza una prueba de concepto para generar una implementación preliminar de los CSWRs que los desarrolladores puedan utilizar como base para sus tareas de implementación y así ahorrar tiempo de trabajo. No se trata de proveer implementaciones que los desarrolladores puedan copiar y pegar sin más, sino que se pretende proveer un punto de partida para el desarrollo que al menos permita comprender en detalle cómo funcionan los CSWRs.

Volviendo a las dificultades para darle la importancia que merece la UX dentro de los métodos ágiles, los desarrollos realizados en este trabajo contribuyen a la sincronización de las prácticas de los expertos en UX y los desarrolladores. En particular, se busca dar soporte para mejorar la experiencia de usuario de una UI, teniendo en cuenta que hasta el momento no existen mecanismos que ante ciertos problemas de UX, les faciliten a los expertos la inspección de posibles soluciones y la evaluación y comparación de éstas para determinar cuál es la que mejor se adecúa a cada situación particular.

1.2. Objetivos

Teniendo en cuenta la necesidad de evaluar la experiencia de usuario en ciclos de desarrollo ágiles y la falta de asistencia para realizar esta tarea, el objetivo general de este trabajo es *proveer un conjunto de métodos y herramientas para dar soporte en la evaluación y solución de problemas de UX durante el desarrollo ágil de las aplicaciones web*. En particular, se pretende facilitarle el trabajo al experto en UX, mediante una serie de mecanismos y herramientas que le ayuden a explorar diferentes alternativas de diseño en una UI, evaluar su funcionamiento, y finalmente seleccionar la solución que más se adecúe al caso para su posterior implementación.

Este objetivo general se descompone en los siguientes objetivos específicos:

- **Generar variantes de diseño.** A partir de una interfaz de usuario desarrollada que

contiene ciertos problemas de UX, se propone el uso de la técnica de refactoring para explorar y analizar posibles soluciones a éstos. El objetivo es que los diseñadores de UX puedan crear alternativas de diseño por su cuenta, es decir, sin requerir de los desarrolladores para implementar los cambios. Para esto, se propone implementar un mecanismo de End-user Development (EUD) [Lieberman et al., 2006] que les permita a los diseñadores aplicar los refactorings de forma asistida, seleccionando los elementos a modificar en la misma UI e ingresando los parámetros adicionales requeridos. Si bien ya existen refactorings alternativos que resuelven un mismo problema de UX, el hecho de utilizar los refactorings como medio para generar cambios de diseño presupone extender el catálogo de refactorings con el objetivo de brindarle al diseñador más de una solución para los distintos problemas.

- **Evaluar y comparar la variantes generadas.** Se propone desarrollar un mecanismo para comparar las variantes de diseño generadas con el propósito de determinar aquella que provee una mejor experiencia a los usuarios. Si bien los refactorings fueron diseñados para mejorar la UX, es necesario evaluar el impacto que produce en los usuarios cada aplicación concreta de un refactoring, para lo cual se pretende desarrollar una métrica que permita evaluar los diseños con una gran cantidad de usuarios reales, por ejemplo a través de pruebas online como los experimentos de A/B testing [Kohavi and Longbotham, 2017].

1.3. Contribución

Los siguientes puntos resumen la contribución concreta de este trabajo:

- Un método de EUD para explorar variantes de diseño utilizando la técnica de refactoring.
- La incorporación de nuevos CSWRs al catálogo desarrollado en trabajos anteriores [Grigera et al., 2017a].
- Un mecanismo de adaptación de estilos que permite personalizar ciertas propiedades estéticas de los cambios de diseño impuestos por los refactorings.
- Una herramienta denominada UX-Painter que implementa los 3 puntos anteriores. Esta herramienta permite crear versiones alternativas de una aplicación web mediante la aplicación asistida de CSWRs. Adicionalmente, UX-Painter da la posibilidad de

generar una implementación preliminar de los refactorings aplicados en cada versión, para facilitarles el trabajo a los desarrolladores en la etapa de implementación.

- La métrica llamada *esfuerzo de interacción* que sirve para evaluar el funcionamiento de los distintos elementos o widgets de la UI que se ven modificados por los refactorings. Los tipos de widget considerados son 6: *campos de texto*, *menús desplegables*, *links*, *botones de radio*, *datepickers* y *menús desplegables de fecha*.
- Un conjunto de *micro-medidas* para cada uno de los 6 tipos de widget que cuantifican diferentes aspectos de la interacción del usuario.
- Modelos de predicción que permiten estimar el esfuerzo de interacción de un widget utilizando como entrada las micro-medidas calculadas automáticamente.
- Una herramienta llamada UX-Analyzer que utiliza los modelos anteriores para calcular y visualizar el esfuerzo de interacción de distintas versiones de una página web. Esta herramienta no solamente permite monitorear el esfuerzo de interacción los widgets incluidos en una página, sino que además proporciona el esfuerzo global de la misma, lo cual facilita la comparación de diseños alternativos.

Estas contribuciones han sido publicadas en revistas de alto impacto y conferencias relevantes en el área, las cuales se listan en la sección 8.3.

1.4. Organización del documento

Más allá de este capítulo introductorio, este trabajo se encuentra estructurado en los siguientes capítulos:

- Capítulo 2: introduce todo el contexto teórico necesario para comprender las contribuciones de este trabajo. Esto incluye entre otras cosas el concepto de UX, sus métodos de evaluación, las propuestas para integrar la UX en los métodos ágiles y la técnica de refactoring de UX. Al mismo tiempo que se describen los conceptos, se analizan trabajos relacionados identificando las diferencias y similitudes con la propuesta de esta tesis.
- Capítulo 3: presenta a grandes rasgos la solución propuesta teniendo en cuenta los 2 objetivos específicos detallados previamente. El capítulo comienza con una descripción más detallada de la motivación de este trabajo y luego se describen las contribuciones realizadas en pos de cumplir con los objetivos.

- Capítulo 4: desarrolla la propuesta del uso de los refactorings de UX como medio para explorar variantes de diseño en la UI real y sin tener que implementar estos cambios. Primero se describe el catálogo completo de CSWRs junto con el mecanismo de adaptación de estilos incorporado, y luego se proporcionan los detalles de UX-Painter. Para finalizar, se pone a esta herramienta en el contexto de un ciclo de desarrollo ágil.

- Capítulo 5: introduce el esfuerzo de interacción como una métrica para evaluar automáticamente el impacto de los refactorings. El desarrollo de esta métrica comienza por la descripción de los tipos de widgets analizados y el proceso de selección de las micro-medidas consideradas para cada uno. Luego se detalla todo el proceso de desarrollo de los modelos de predicción, que incluye la recolección de datos necesarios para el entrenamiento y la implementación misma de los modelos. Por último, se presentan los resultados obtenidos y un análisis de la importancia que cobra cada micro-medida en el proceso de predicción.

- Capítulo 6: describe la herramienta UX-Analyzer que permite calcular el esfuerzo de interacción de diferentes páginas web, incluyendo ejemplos de las diferentes funcionalidades que ofrece.

- Capítulo 7: presenta las distintas evaluaciones realizadas. Por un lado, se describen las evaluaciones relacionadas al método para generar variantes de diseño, que incluye una evaluación del método en sí y una validación de usabilidad de UX-Painter. Por el otro lado, se detalla la evaluación del esfuerzo de interacción que se basa en la comparación con otras métricas similares que sirven al mismo propósito.

- Capítulo 8: presenta un repaso de la contribución analizando los logros obtenidos y luego se describen posibles trabajos futuros.

Capítulo 2

Marco teórico y trabajo relacionado

En este capítulo se describen los conceptos necesarios para comprender la contribución de este trabajo, al mismo tiempo que se repasan los trabajos más relevantes en cada una de las áreas de investigación involucradas en esta propuesta. Primero se explica el concepto de UX y las nociones básicas de los métodos ágiles, que incluyen el origen de la técnica de refactoring y los intentos por integrar la UX como parte del desarrollo en estos métodos. Luego se presentan las cuestiones relacionadas a la generación de cambios de diseño, empezando por el punto central de este trabajo que son los refactorings de UI, siguiendo por las nociones de adaptación web y End-user development, y para terminar se hace un análisis de las herramientas diseño que pueden compararse con esta propuesta. Finalmente se describen los métodos de evaluación de la UX, el origen del esfuerzo de interacción y las nociones de aprendizaje automático.

2.1. Experiencia de Usuario

En la actualidad el término *experiencia de usuario* está muy de moda. Si bien es difícil encontrarle una definición precisa a este concepto, a grandes rasgos se podría decir que tiene que ver con cómo se siente una persona al interactuar con un producto o servicio. Específicamente dentro del mundo de la web, la importancia que se le da hoy en día a la UX en gran parte se debe a la gran proliferación de aplicaciones, lo cual de cierto modo obliga a las empresas a proveer experiencias positivas para asegurarse la fidelidad de los usuarios a su marca o producto. Por su parte los usuarios demandan cada vez más productos que

no solamente sean fáciles de usar, sino que además les resulten atractivos y les permitan cumplir con sus necesidades con altos niveles de satisfacción.

Yendo a una definición formal, la norma ISO 9241-210 define a la experiencia de usuario como las *percepciones y respuestas de los usuarios que resultan del uso y/o anticipado uso de un sistema, producto o servicio* [ISO, 2019]. Según esta definición, UX es un concepto muy amplio que incluye las reacciones emocionales, cognitivas y físicas de los usuarios que surgen antes, durante y después del uso de un sistema.

Desde un principio, la investigación en el área de HCI se ha centrado en el estudio de los aspectos instrumentales de un sistema [Hassenzahl and Tractinsky, 2006]. En particular, la atención ha estado puesta en la **usabilidad**, utilizando el concepto de tarea para determinar cuan efectiva y eficiente es la interacción de un usuario con un sistema dado. En este sentido, el análisis de la usabilidad consiste básicamente en la aplicación de la psicología cognitiva para mejorar ciertos indicadores que reflejan la performance del usuario con el sistema.

Más allá de la importancia que tiene la usabilidad de un sistema, hace unos años que se reconoce que este concepto ofrece una visión incompleta de la interacción del usuario con un sistema, porque no tiene en cuenta los aspectos hedónicos que van más allá de las cuestiones instrumentales, como por ejemplo el disfrute y las propiedades estéticas de los diseños [Hassenzahl et al., 2001]. Es por este motivo que el término UX se utiliza como un concepto integrador que abarca tanto los aspectos hedónicos, que hacen referencia a las preferencias y emociones de los usuarios, como también los aspectos instrumentales, que involucran la funcionalidad y performance del sistema. Teniendo en cuenta esto, en otras palabras se puede decir que la UX es una consecuencia del estado interno del usuario (sus expectativas, necesidades, humor, etc.), las características del sistema (complejidad, funcionalidad, usabilidad) y el contexto en el cual la interacción tiene lugar [Hassenzahl and Tractinsky, 2006].

Otras definiciones de UX también comparten esta visión holística que incluye la usabilidad junto con otras cuestiones subjetivas [Law et al., 2009, Vermeeren et al., 2010]. De hecho Vermeeren et al. sostienen que aunque la experiencia de usuario puede considerarse como parte del componente subjetivo de usabilidad (la satisfacción), la noción de UX tiene en cuenta otras cualidades subjetivas que no forman parte de la satisfacción.

Considerando la visión general que proporciona el concepto de UX, se lo utiliza a lo largo de este trabajo para no limitarse únicamente a cuestiones de usabilidad.

2.2. Métodos ágiles

Actualmente los métodos ágiles son ampliamente utilizados en la industria del software para el desarrollo de aplicaciones. Estos métodos se basan fundamentalmente en realizar entregas incrementales en períodos de tiempo cortos, con el objetivo de satisfacer al *cliente* mediante la entrega continua de software *funcional* con valor [Rubin, 2012]. En esta sección se hace un repaso de la técnica de refactoring, una práctica común dentro de los métodos ágiles para garantizar la calidad del código desarrollado, y luego se describen algunos trabajos que destacan la importancia de introducir la evaluación de la UX en los métodos ágiles con el objetivo de no sólo desarrollar software que sea funcional, sino que además provea una buena experiencia de usuario.

2.2.1. Refactoring de código y deuda técnica

El concepto de refactoring fue propuesto inicialmente por Opdyke para referirse a las transformaciones de código cuyo objetivo era mejorar el diseño del mismo, preservando la funcionalidad del sistema [Opdyke, 1992]. Esta técnica se popularizó más tarde cuando Fowler desarrolló un catálogo de refactorings, que consistió en una serie de transformaciones aplicables a código orientado a objetos apuntadas a mejorar aspectos como legibilidad, extensibilidad, entre otros [Fowler et al., 1999]. Estas transformaciones se aplican ante la presencia de potenciales deficiencias de diseño que en la jerga de refactoring se llaman *bad smells*. Un bad smell señala un aspecto del código que puede mejorarse. Por ejemplo, el bad smell "Duplicated Codeïndica que hay una porción de código que está repetida. Como solución a este smell, se puede aplicar el refactoring "Extract Method"para crear un nuevo método con el código repetido utilizando un nombre de método que describa su propósito.

La técnica de refactoring permite ir mejorando de forma incremental y sistemática el código de una aplicación. Esto la ha convertido en una práctica esencial dentro de los métodos ágiles que están centrados en realizar entregas continuas de software funcionando. La presión constante por cumplir con las expectativas del cliente en tiempo y forma, muchas veces lleva a tomar decisiones (consciente o inconscientemente) que van en detrimento de la calidad del código. Ward Cunningham creó el concepto de *deuda técnica* como una metáfora que compara los problemas de código que se dejan sin solucionar con la acumulación de una deuda financiera [Cunningham, 1992]. Así como una deuda financiera se incrementa con el paso del tiempo, las deficiencias del diseño del código se vuelven más complejas de resolver a medida que avanza el desarrollo. Y si bien muchas veces la deuda se adquiere conscientemente para obtener un beneficio inmediato como podría ser la incorporación de

una nueva funcionalidad, en algún momento debe "pagarse"solucionando los problemas del código para no empeorar su calidad.

Teniendo en cuenta la tendencia de los métodos ágiles a generar deuda técnica, el refactoring es el principal medio para mantenerla controlada, dado que los desarrolladores pueden primero enfocarse en codificar una funcionalidad, y luego incrementalmente mejorar la calidad del código sin alterar su comportamiento obtenido, en pequeños pasos seguros. El alcance de esta técnica se expandió rápidamente hacia otros paradigmas de programación y más allá de la mejora de atributos de calidad interna de código, para mejorar aspectos externos (visibles) del software, como seguridad en bases de datos [Ambler and Sadalage, 2006] y navegabilidad en aplicaciones web [Cabot and Gómez, 2008].

2.2.2. Integración de métodos ágiles con métodos de evaluación de UX

Si bien los métodos ágiles permiten desarrollar software rápidamente atendiendo las necesidades cambiantes de los clientes, la experiencia de usuario no es un atributo esencial a considerar durante el desarrollo. Por otro lado, la gran relevancia que ha adquirido la UX en los últimos años, puso la atención en procesos como el Diseño Centrado en el Usuario (DCU) [Garrett, 2002] que combina una variedad de técnicas de investigación y diseño para determinar las necesidades de los *usuarios* y así crear productos altamente *usables* y *accesibles*.

Tanto los métodos ágiles como DCU tienen por objetivo crear productos de calidad, pero cada uno lo hace desde una perspectiva diferente. DCU pone el foco en el usuario final del producto mientras que los métodos ágiles enfatizan en la colaboración con el cliente, que es quien establece los requerimientos del producto pero no necesariamente es el usuario del mismo. En el manifesto ágil se habla de software funcionando como medida de progreso, pero a diferencia de DCU no es prioridad proveer una experiencia óptima a los usuarios finales. Con respecto a las tareas de diseño, en DCU se suele reservar un tiempo considerable para realizarlas antes de comenzar el desarrollo, mientras que en un ciclo ágil, el diseño se hace bajo demanda debido a la escasez de tiempo. Por último, en los métodos ágiles se utilizan las pruebas de unidad y de aceptación para evaluar los desarrollos, pero generalmente no se realizan pruebas con usuarios finales para garantizar una buena experiencia de usuario.

A partir de estas diferencias, han surgido varios intentos de integrar los dos enfoques [Silva et al., 2011, Jurca et al., 2014, Brhel et al., 2015]. Estos estudios coinciden en que generalmente es necesario incorporar una etapa de diseño temprano no muy extensa (llamada *Little Design Upfront*) para generar una UI consistente, que puede iniciarse en el *Sprint 0* antes que comience el desarrollo. Otra práctica común es la denominada *One Sprint Ahead*

que establece que el equipo de diseño debe trabajar al menos una o dos iteraciones adelante del resto del equipo, aunque también hay propuestas de equipos combinados trabajando en la UI y en la implementación en el mismo ciclo [Kuusinen, 2016]. Sin embargo, hasta el momento no hay un acuerdo común acerca de cómo llevar adelante estas prácticas.

Un aspecto fundamental para lograr una integración entre los métodos ágiles y DCU es la colaboración entre el equipo de desarrollo y el equipo de UX [Silva et al., 2013]. Uno de los medios que han sido propuestos para facilitar la comunicación entre ambos equipos son los *artefactos de software* [Brhel et al., 2015]. Esta idea de comunicación mediante artefactos va en línea con el principio ágil que desalienta la documentación excesiva y que establece que el conocimiento se intercambia principalmente mediante las interacciones entre los miembros del equipo. Con respecto a los tipos de artefactos a utilizar, Da Silva et al. enfatizan en la necesidad de *artefactos digitales*, teniendo en cuenta que los equipos de desarrollo distribuidos geográficamente son cada vez más comunes [Silva et al., 2018].

Siguiendo con la comunicación mediante artefactos, Garcia et al. realizaron un estudio netnográfico para determinar cuáles son los artefactos más usados por los equipos de desarrollo [Garcia et al., 2019]. Los resultados arrojaron que las *historias de usuario* y los *wireframes* fueron los artefactos más mencionados. Las historias de usuario se usan en la etapa de planificación de un sprint para entender claramente qué es lo que se tiene que implementar y para estimar el esfuerzo de implementación. En cuanto a los wireframes, se utilizan en la etapa de diseño para ilustrar la estructura de la UI y en la planificación como soporte a las historias de usuario. Más allá de los artefactos más usados, la discusión de los resultados pone en evidencia la **falta de artefactos destinados a evaluar y mejorar la experiencia de usuario**. Entre los artefactos mencionados también aparecen los *prototipos*, que si bien pueden ser útiles para realizar evaluaciones de UX tempranas, no son apropiados para etapas posteriores en las que ya se cuenta con una UI desarrollada, y se debe dar solución a ciertos problemas de UX generados durante el uso de la misma.

Esta falta de soporte para resolver problemas de experiencia de usuario hace que la UX sea dejada de lado, lo que genera que con el tiempo estos problemas se acumulen, y esta acumulación puede derivar en el fracaso de la aplicación o producto.

2.3. Refactorings de UI

Como se mencionó anteriormente, la gran adopción del refactoring hizo que la técnica se propague a otros artefactos de software y con propósitos variados que van más allá de mejoras internas al código. De esta manera, surgió la idea de utilizar el refactoring para

mejorar la usabilidad de las aplicaciones web, dando lugar al concepto de *usability refactoring* [Garrido et al., 2007]. En un principio, el refactoring se planteó sobre los modelos de presentación y navegación de una aplicación web, para mejorar las aplicaciones que se derivan de estos [Garrido et al., 2011]. Con los refactorings propuestos, los desarrolladores podían lograr mejoras en la usabilidad de la aplicación resultante, como por ejemplo una mejor distribución de contenidos en pantalla entre páginas, una mejor estructura de navegación, entre otros. Estos refactorings surgieron como solución a un conjunto de malos olores (*usability smells*), que indican posibles problemas de usabilidad que pueden mejorarse.

Más tarde, el concepto de usability refactoring se empezó a utilizar para describir transformaciones en la interfaz de usuario de una aplicación web, que pueden aplicarse del lado del cliente [Garrido et al., 2013]. Allí, se definen los *Client-side web refactorings* como scripts predefinidos que modifican el DOM (Document Object Model) de una página web con el objetivo de mejorar la usabilidad pero sin alterar la funcionalidad de la misma. Este cambio en la idea de usability refactoring hizo que se desarrollen nuevos usability smells y nuevos refactorings. Además, con el objetivo de facilitar la evaluación de usabilidad, se desarrollaron herramientas para detectar automáticamente usability smells y aplicar refactorings para solucionarlos.

Usability Smells Finder permite identificar los usability smells haciendo un análisis y filtrado de los eventos de interacción generados en el navegador [Grigera et al., 2017a]. Un ejemplo concreto de smell es *Unformatted Input*, que ocurre cuando múltiples usuarios ingresan en un campo de texto libre valores que tienen un formato determinado. Kobold es un servicio web que utiliza Usability Smells Finder para reportar los smells de una aplicación web y para cada uno de estos sugiere uno o más refactorings como solución, que en ciertos casos se aplican automáticamente mediante los CSWRs [Grigera et al., 2017b]. Siguiendo con el ejemplo anterior y suponiendo que el valor a ingresar corresponde a una fecha, hay dos usability refactorings que solucionan Unformatted Input: *Add Datepicker*, que agrega en el campo de texto un calendario para elegir la fecha, y *Date Input into Selects* que directamente reemplaza el campo de texto por tres menús desplegables para elegir día, mes y año. Para aplicar un refactoring, la herramienta en algunos casos le solicita al usuario ciertos parámetros adicionales y luego inyecta el script del refactoring en la aplicación. A partir de ese momento, todos los usuarios que ingresen a la aplicación verán los cambios realizados por este refactoring.

La principal ventaja de Kobold es que permite aplicar mejoras de usabilidad automáticamente, lo cual puede ser de gran utilidad para equipos que no cuenten con la ayuda de un experto en UX para descubrir problemas y solucionarlos. Pero por otro lado, el inconve-

niente que presenta esta herramienta y que se pretende abordar en este trabajo, es que los refactorings se aplican directamente en la UI que se encuentra en producción sin antes poder evaluar el impacto de las modificaciones. Esto es fundamental no solamente para comprobar que un refactoring mejora la experiencia de usuario, sino también para poder determinar ante soluciones alternativas como las del ejemplo anterior, cuál es la más efectiva en cada situación particular.

Es importante mencionar que a lo largo de este trabajo se utiliza el concepto de *refactorings de UX* [Garrido et al., 2017], dado que los refactorings pueden utilizarse para explorar cambios de diseño que no tengan ver estrictamente con cuestiones de usabilidad. De acuerdo a la definición provista anteriormente, UX es un concepto más general que engloba no solamente usabilidad, sino también aspectos hedónicos.

2.4. Adaptación web

Hace tiempo que el navegador se viene utilizando como una plataforma para realizar modificaciones en una interfaz de usuario. De hecho, actualmente los navegadores proveen herramientas para facilitar el desarrollo y testing de una aplicación. Por ejemplo, las herramientas de desarrollo ayudan a los desarrolladores y diseñadores a ver, inspeccionar, depurar y modificar una UI "on the fly", aunque para esto se requieren conocimientos de programación y los cambios generados se pierden al recargar la página. Por otro lado, los navegadores también dan la posibilidad de modificar la apariencia y el comportamiento de las aplicaciones web mediante extensiones web o *addons* instalables.

El hecho de poder modificar una aplicación externamente a través del navegador, dio lugar a diferentes mecanismos de adaptación web. Uno de estos mecanismos es la *aumentación web* (web augmentation), que consiste en darle la posibilidad a los usuarios finales de incorporar o extender funcionalidades de aplicaciones existentes sin depender de sus desarrolladores [Aldalur et al., 2017]. Ejemplos concretos de herramientas de aumentación web son aquellas que permiten personalizar el contenido de una página web [Díaz et al., 2016], y las que tienen por objetivo integrar contenido de diferentes aplicaciones web, formando lo que se conoce técnicamente como *mashups* [Wong and Hong, 2007, Ghiani et al., 2016].

Teniendo en cuenta que Díaz et al. definen a la aumentación web como *el proceso de agregar o modificar el contenido, diseño o navegación de una web existente con el objetivo de personalizar la experiencia de usuario* [Díaz et al., 2013], el uso de los CSWRs para modificar una página web externamente puede considerarse como un caso de adaptación web. La diferencia con los trabajos antes mencionados radica en que el propósito de los

CSWRs es mejorar la interacción del usuario en lugar de incorporar nueva funcionalidad.

2.5. End-user development

El término End-user development ha sido ampliamente utilizado en las últimas décadas para referirse a un conjunto de técnicas y herramientas que les permiten a los usuarios finales, que no necesariamente cuentan con conocimientos de programación, desarrollar y adaptar aplicaciones por su cuenta [Lieberman et al., 2006]. Este concepto surgió a partir de los diferentes requerimientos y necesidades que poseen los usuarios de las aplicaciones que últimamente son de uso masivo. Lógicamente, parte de la solución para atender esta variedad de requerimientos está en involucrar a los usuarios en etapas tempranas de desarrollo, como lo hace por ejemplo DCU. Pero lo cierto es que los requerimientos de los usuarios muchas veces son difíciles de determinar de antemano y cambian muy rápidamente, por lo cual EUD aparece como una opción para que los usuarios puedan adaptar un sistema a sus necesidades.

La gran mayoría de las propuestas de EUD están destinadas a usuarios que no tienen ningún tipo de vínculo con el desarrollo de aplicaciones. Estos trabajos incluyen una plataforma para involucrar a los usuarios en el diseño de aplicaciones web [Nebeling et al., 2012], un ecosistema para el desarrollo de aplicaciones móviles multi-plataforma [Zhai et al., 2016] y la incorporación de servicios de búsqueda en páginas web [Bosetti et al., 2022].

Sin embargo, en este se trabajo se adhiere a la definición de EUD provista por Ko et al. que establece que End-user development se trata de *programar algo principalmente para uso personal más que para el uso público* [Ko et al., 2011]. Entre las implicaciones que tiene está definición, es importante destacar que el término "usuario final"no se refiere a las habilidades del usuario. De hecho en [Aldalur et al., 2017] se da como ejemplo de una actividad de EUD a un desarrollador que implementa código para satisfacer una necesidad personal, como podría ser una herramienta que facilite el diagnóstico de errores.

Teniendo en cuenta lo anterior, el método desarrollado en este trabajo para generar variantes de diseño se enmarca dentro de una propuesta de EUD, porque sirve como soporte tanto a los expertos en UX en sus tareas de inspección y evaluación de modificaciones a una UI, como también a los desarrolladores que posteriormente deben implementar estos cambios.

2.6. Herramientas para explorar cambios de diseño

Los prototipos son artefactos ampliamente utilizados en la etapa de diseño de una aplicación para acordar entre los miembros de un equipo de desarrollo cómo debe verse la interfaz de usuario, y también para explorar diferentes ideas de diseño sin incurrir en costos de implementación. Tanto los prototipos de baja fidelidad, que modelan aspectos generales de la interacción sin entrar en detalles, como los de alta fidelidad que representan cuestiones más precisas, han sido propuestos para realizar inspecciones y pruebas de usabilidad con usuarios [Silva et al., 2011]. Los prototipos de baja fidelidad sirven para verificar la estructura general de la UI considerando aspectos técnicos [Garcia et al., 2019]. En cuanto a los de alta fidelidad, existen herramientas comerciales como Adobe XD[1] o InVision[2] que proveen entornos para crear diseños sofisticados que incluyen interacciones y animaciones, lo cual permite realizar evaluaciones de UX en un contexto más cercano al de la aplicación real. Sin embargo, por más que estos prototipos simulen la interacción, las evaluaciones en la UI real siguen siendo fundamentales debido a que pueden surgir problemas de UX no reproducibles en estos artefactos. Por este motivo, parte del objetivo de este trabajo es el mismo que el de las herramientas de prototipado, que es el de poder explorar rápidamente variantes de diseño con el mínimo esfuerzo. La diferencia está en que en este trabajo se propone hacer esta exploración en la misma aplicación en lugar de generar un nuevo artefacto.

Por otro lado, las herramientas para realizar experimentos de A/B testing ofrecen editores para generar las variantes a evaluar que trabajan directamente sobre la UI real. La limitación de estos editores es que están diseñados para realizar cambios en las propiedades estéticas de una página tales como colores, fuentes y etiquetas, por lo cual para lograr cambios de diseño más complejos se requiere editar el código fuente. A diferencia de esto, las soluciones predefinidas que se proponen en este trabajo para generar la variantes de diseño están más enfocadas en resolver problemas de interacción, como agregar validación a un formulario, reemplazar un widget por otro, etc.

Dentro de los trabajos académicos, es importante mencionar a Supple, una herramienta que genera automáticamente una UI óptima para un dispositivo particular y ciertas necesidades del usuario [Gajos et al., 2010]. Supple fue diseñada para que usuarios finales puedan crear una UI personalizada, para lo cual define un modelo abstracto incluyendo ciertas restricciones del usuario y del dispositivo que posteriormente se utilizan como entrada en un algoritmo de optimización. En este sentido, la herramienta es apropiada para atender

[1] https://www.adobe.com/la/products/xd
[2] https://www.invisionapp.com

necesidades especiales de los usuarios, ya que incluso fue evaluada con usuarios con ciertas discapacidades motoras. Más allá de que Supple tiene un objetivo diferente al de este trabajo, el mecanismo que se utiliza para generar los distintos diseños es similar al de los refactorings en cuanto a que se basa en el uso de elementos o widgets alternativos de una UI, asegurando que la funcionalidad de la interfaz no se ve alterada.

2.7. Métodos de evaluación de UX

UX es un concepto abstracto que no puede evaluarse directamente, sino que se deben seleccionar atributos que influyan en la experiencia de usuario y que se puedan medir de alguna forma concreta. Dentro los aspectos hedónicos, ejemplos de estos atributos son la complejidad visual percibida (visual complexity) y las propiedades estéticas de una aplicación web (web aesthetics) [Michailidou et al., 2008, Reinecke et al., 2013]. Por el lado de los aspectos instrumentales, la evaluación generalmente se centra en los atributos de usabilidad que son eficacia, eficiencia y satisfacción [ISO, 2018].

Una clasificación ampliamente usada para los métodos de evaluación de UX es la que divide los **métodos de inspección** y los **métodos empíricos** [Fernandez et al., 2011]. Los primeros son realizados por expertos en UX y consisten básicamente en analizar un artefacto para determinar si cumple o no con determinadas heurísticas predefinidas. Ejemplo de estas heurísticas son las propuestas por Nielsen [Nielsen, 1994]. Con respecto a los métodos empíricos, se basan en capturar y analizar datos de usuarios reales ya sea en la aplicación o en un prototipo, con el objetivo de identificar problemas durante la interacción. El ejemplo más común de estos métodos son las pruebas con usuarios (user tests) [Rubin and Chisnell, 2008]. La ventaja de los métodos de inspección es que se pueden realizar tempranamente en artefactos como mockups o prototipos en papel, y la principal limitación es que los resultados dependen de la calidad de las heurísticas y del evaluador que las analice. Por otro lado, los métodos empíricos tienen la ventaja de poder considerar diferentes tipos de usuarios finales, pero para poder realizarse requieren que la aplicación esté implementada, por lo cual son más costosos que los métodos de inspección y se suelen hacer en las últimas etapas de desarrollo.

En este trabajo el foco está puesto en los métodos empíricos realizables en aplicaciones web, en particular en aquellos métodos automáticos que permiten abaratar los costos de la evaluación de la UX. Bakaev et al. [Bakaev et al., 2017] clasifican los métodos automáticos en tres categorías: los *basados en métricas* (metric-based) que ayudan a los expertos en UX en el análisis estático de páginas web, los *basados en interacción* (interaction-based) que

analizan datos de usuarios reales, y finalmente los *basados en modelos* que utilizan modelos de predicción o de simulación de usuarios. En esta tesis se propone una métrica para evaluar la UX y su utilización en un método de evaluación que puede considerarse una combinación de las últimas dos categorías, ya que requiere capturar los eventos de interacción de usuario para calcular ciertas micro-medidas que posteriormente se usan como entrada en modelos de predicción. Los métodos basados en métricas están centrados en aspectos de la UX que pueden ser analizados estáticamente (sin requerir de la interacción de usuarios), como las propiedades estéticas o la complejidad visual de una página web. Ejemplos de este método son los trabajos propuestos por Dingli & Mifsud [Dingli and Mifsud, 2011] y Oulasvirta et al. [Oulasvirta et al., 2018]. El primer trabajo presenta USEful, un framework de evaluación de usabilidad web que sirve como soporte a los métodos de inspección, específicamente para el chequeo de ciertos estándares en el código HTML. Por otro lado, Oulasvirta et al. desarrollaron AIM, un servicio web que sirve para evaluar una página web utilizando un conjunto de métricas que miden diferentes propiedades de una UI como simetría (symmetry), colores (colorfulness) y desorden visual (visual clutter).

Con respecto a los métodos basados en interacción, existen herramientas que capturan y analizan logs de interacción durante pruebas de usuario remotas y mediante diferentes técnicas de visualización de datos proveen información útil para identificar problemas de UX, por ejemplo utilizando líneas de tiempo [Burzacca and Paternò, 2013, Paternò et al., 2017] y grafos de uso [de Santana and Baranauskas, 2015]. También hay trabajos que relacionan el movimiento del mouse con atención visual [Arroyo et al., 2006], con el seguimiento de la vista (eye tracking) [Navalpakkam and Churchill, 2012] y con la dificultad para responder cuestionarios [Horwitz et al., 2017]. La particularidad que tienen estos trabajos es que proveen feedback de una o varias páginas web, pero no permiten localizar con precisión dónde los usuarios experimentan problemas. Esta limitación se aborda en este trabajo mediante la definición de una métrica que determina el esfuerzo requerido por los usuarios para interactuar con los diferente elementos de una UI.

Dentro de los métodos basados en interacción también se pueden incluir los experimentos online como las pruebas de A/B testing y sus variantes [Kohavi and Longbotham, 2017]. Estas pruebas consisten en presentarle a los usuarios versiones alternativas de una página web para determinar cuál se desempeña mejor en base a una métrica objetivo. El principal punto a favor de estos experimentos es que permiten evaluar una página web en su contexto real, con múltiples usuarios, y sin que estos sean conscientes que están siendo evaluados. Pero por otro lado, las métricas objetivo que se utilizan suelen medir aspectos relacionados al desempeño de una organización o negocio, como por ejemplo la cantidad de suscripciones

de usuario o la cantidad de clicks en un botón de compra, que si bien se pueden ver afectadas por la experiencia de los usuarios, no son indicadores directos de la UX [Nielsen, 2005]. Otra desventaja de estas métricas relacionada con lo anterior es que no permiten determinar el por qué de los resultados obtenidos; al finalizar el experimento sólo se sabe que más usuarios hicieron una determinada acción en un diseño que en otro, pero no hay ningún indicio acerca de qué fue lo generó este comportamiento. Como solución a estas limitaciones, el esfuerzo de interacción definido en este trabajo se propone como una métrica que puede utilizarse en este tipo de experimentos como un indicador de la UX, con la ventaja adicional que permite localizar dónde los usuarios experimentan problemas de interacción.

Volviendo a la clasificación de los métodos de evaluación, en la categoría de los métodos basados en modelos han surgido diferentes propuestas para predecir la interacción del usuario. Varios años atrás, modelos como la ley de Fitts [Fitts, 1954] y KLM (Keystroke Level Model) [Card et al., 1980] fueron desarrollados para estimar el tiempo requerido de una tarea sin requerir la participación de usuarios. Sin embargo, la desventaja que tienen estos métodos es que fueron diseñados para capturar un único aspecto de la interacción de usuario en forma aislada, por lo cual es difícil usarlos para hacer predicciones en una UI real [Li et al., 2018]. Por otro lado, la introducción de la inteligencia artificial en HCI permitió un gran avance en la automatización de la evaluación de la UX. Los modelos de aprendizaje automático sirven para encontrar patrones complejos en los datos, que serían muy difíciles de detectar analíticamente. Este tipo de modelos recientemente han sido utilizados para predecir la complejidad visual percibida de una página web, utilizando métricas obtenidas mediante el análisis estático de la página objetivo [Michailidou et al., 2021], y también usando modelos de aprendizaje profundo [Dou et al., 2019].

En cuanto a los modelos de predicción que tienen en cuenta la interacción del usuario, hay trabajos que estiman la performance de usuarios en la selección de ítems de menús. Bailly et al. desarrollaron un modelo matemático para calcular el tiempo de selección en menús lineales [Bailly et al., 2014], mientras que Li et al. utilizan una red neural recurrente para estimar la performance de los usuarios en una secuencia de tareas [Li et al., 2018]. Las dos propuestas tienen como objetivo predecir aspectos de la interacción sin necesitar de usuarios, pero están limitadas a una interacción muy específica que es la selección de una opción de menú. Además, hay aspectos de la performance de los usuarios que dependen del contexto de uso de cada usuario, que no se consideran en estos modelos. Otro trabajo más parecido al de esta propuesta es el de Speicher et al. [Speicher et al., 2014], en el que también se propone el uso de los eventos de interacción de usuario para hacer predicciones; la diferencia radica en que las métricas que se proponen allí están muy atadas al diseño de

la UI de la página destino, por lo cual es necesario desarrollar un modelo diferente para cada grupo de páginas que comparten un diseño común. En cambio, el análisis de una UI a través de sus widgets que se propone en esta tesis es independiente del tipo de aplicación web objetivo, porque los widgets analizados son componentes estándar que se utilizan en cualquier aplicación web.

2.8. Interactividad y carga cognitiva

Janlert y Solterman definieron el concepto de "interactividadçomo *interacción en acción* [Janlert and Stolterman, 2017]. La interactividad de un sistema puede fluctuar de un momento a otro durante su uso, por lo cual es importante poder medirla para mejorar la UX del mismo. Una forma de medir la interactividad es analizar el tiempo requerido para interactuar con una UI [Janlert and Stolterman, 2017], que incluye tanto las acciones explícitamente dirigidas por el usuario hacia la interfaz, como también aquellas acciones que pueden considerarse parte de lo que se conoce como "interacción implícita". La interacción implícita se refiere a aquellas acciones que ocurren inconsciente y automáticamente durante el uso de una UI, como por ejemplo movimientos del cursor alrededor de un elemento, scroll de la pantalla, entre otros [Atterer et al., 2006].

Existen trabajos que muestran que estos patrones de interacción implícita, que pueden recolectarse analizando el uso de un sistema, reflejan en cierto punto el esfuerzo mental y la *carga cognitiva* experimentada por el usuario [Gütl et al., 2005, Chen et al., 2012]. De hecho, dentro del amplio abanico de HCI, existen diversos estudios acerca de la carga cognitiva debido a la escasa capacidad de la memoria a corto plazo de los usuarios [Baddeley, 1979].

En la literatura, existen tres tipos de carga cognitiva. La *intrínseca* que tiene que ver con la complejidad en sí de la tarea y la expertise del usuario, la *extraña* que se da cuando el material se presenta de una forma inadecuada o cuando el usuario debe realizar actividades consideradas irrelevantes, y por último la *germánica* que está vinculada con los procesos para generar aprendizaje [DeLeeuw and Mayer, 2008]. De acuerdo a Hollender et al. la carga cognitiva inducida por el uso de un sistema puede considerarse un componente de la carga cognitiva extraña [Hollender et al., 2010], que se ve influenciada por el diseño de la experiencia de usuario del mismo sistema. Las acciones del usuario durante la interacción como la voz, gestos y movimiento del mouse constituyen comportamientos observables que pueden considerarse indicadores de la carga cognitiva [Chen et al., 2012, Chen et al., 2016].

El esfuerzo físico y mental que hacen los usuarios para interactuar con una UI, tomados en forma conjunta pueden asociarse con un "costo de interacciónçon esa interfaz

[Budiu, 2013]. Ese costo en este trabajo se denomina esfuerzo de interacción, y está inspirado en los trabajos de interactividad y carga cognitiva extraña que permiten interpretar lo que que se pretende medir. La particularidad adicional que tiene el esfuerzo de interacción es que se calcula sobre los widgets individuales de una UI.

2.9. Aprendizaje automático

El aprendizaje automático (Machine Learning) es un área de la inteligencia artificial (AI) que se centra en el desarrollo de programas que aprenden automáticamente. Un programa que aprende automáticamente es aquel que no se programa explícitamente para resolver un problema, sino que utiliza un algoritmo de aprendizaje en base a datos de ejemplo del problema para generar un modelo del mismo. Con ese modelo, se pueden analizar ejemplos del problema general y realizar predicciones con cierto grado de error.

En general, las técnicas de aprendizaje automático se aplican a problemas donde no se conoce una solución algorítmica definitiva o ésta tiene características indeseables, como por ejemplo ser computacionalmente demandante, requerir programación explícita para adaptarse a cambios o requerir intervención manual. Entre las aplicaciones típicas del aprendizaje automático se puede mencionar el reconocimiento de voz, de imágenes, recomendaciones de productos, entre otros. La particularidad que tienen este tipo de tareas es que es muy difícil identificar reglas para realizarlas explícitamente. Por este motivo, los algoritmos de aprendizaje automático se basan en identificar patrones en los datos de ejemplo que de algún modo permitan emular el razonamiento humano.

Un proceso de aprendizaje automático se suele dividir en 2 grandes etapas. Una es la etapa de *entrenamiento*, en la que se genera el modelo de predicción analizando los datos de ejemplo, y la etapa de *predicción* en la que se hace uso del modelo generado para hacer predicciones a partir de datos nuevos. Se podría decir que la etapa de entrenamiento es el corazón del aprendizaje automático; allí es donde se aplica toda la ingeniería con el objetivo de generar un modelo con el mínimo margen de error posible.

Si bien existen varias clasificaciones de las técnicas de aprendizaje automático, la más significativa es la que distingue entre aprendizaje *supervisado* y *no supervisado*. Los algoritmos supervisados son aquellos en los que se conoce de antemano el resultado que se busca aprender para los datos de ejemplo del problema. Por ejemplo, si se quiere generar un modelo para reconocer animales en imágenes, el aprendizaje es supervisado si para cada imagen de ejemplo se sabe qué animal se encuentra en ella. De esta manera, en la etapa de entrenamiento se utiliza cada imagen de ejemplo junto con una *etiqueta* que indica el resultado

esperado de la predicción. Por otra parte, en el aprendizaje no supervisado los datos no se encuentran etiquetados, por lo cual los algoritmos buscan aprender relaciones entre los datos sin guiarse por ningún resultado esperado. En este trabajo se utiliza un algoritmo de aprendizaje automático porque se busca predecir el esfuerzo de interacción, un puntaje que es asignado por los expertos en UX. Para esto se realizó un proceso de recolección de datos en el que se capturaron interacciones de usuario de ejemplo, que luego fueron etiquetadas por un grupo de expertos en UX.

A su vez dentro del aprendizaje supervisado, existen 2 tipos de algoritmos que dependen de la tarea que se pretende realizar. Los algoritmos de *clasificación*, como su nombre lo indica, permiten clasificar datos dentro de un conjunto de categorías o clases predefinidas. El ejemplo anterior del reconocimiento de animales en imágenes es una tarea de clasificación, en la cual cada animal corresponde a una clase. Estas clases se definen en las etiquetas de cada dato de ejemplo y el resultado del modelo es siempre alguno de estos valores. Por otra parte, los algoritmos de *regresión* se utilizan para predecir valores numéricos que no están sujetos a clases predefinidas, por lo cual los valores predecidos pueden ser diferentes a las etiquetas. Ejemplos de regresión son la predicción del valor bitcoin [McNally et al., 2018] y la estimación del valor de inmuebles [Lim et al., 2016]. La predicción del esfuerzo de interacción es una tarea de regresión porque se trata de estimar un puntaje númerico que se encuentra entre 1 y 4.

2.9.1. Árboles de decisión

Dentro de las categorías descriptas anteriormente, existen diferentes algoritmos para realizar tareas de aprendizaje automático. Entre los más utilizados se encuentran las redes neuronales, las máquinas de soporte vectorial y los árboles de decisión. En esta sección se describen los conceptos básicos de los árboles de decisión, debido a que fue el algoritmo utilizado para predecir el esfuerzo de interacción.

Un árbol de decisión es un algoritmo de aprendizaje supervisado que se utiliza para tareas de clasificación y de regresión [Breiman et al., 1984]. Tiene una estructura de árbol jerárquica, que consta de un nodo raíz, ramas, nodos internos y nodos hoja. La Figura 2.1 muestra un ejemplo de árbol de decisión. Allí se puede apreciar que el nodo raíz no tiene ramas entrantes, las ramas salientes del nodo raíz alimentan los nodos internos, y los nodos hoja representan todos los resultados posibles dentro del conjunto de datos.

El árbol se genera durante la etapa de entrenamiento mediante la división del conjunto de datos de ejemplo. Esta división es un proceso recursivo, en el cual en cada paso se divide al conjunto de datos de un nodo en base al atributo que proporciona la mejor división.

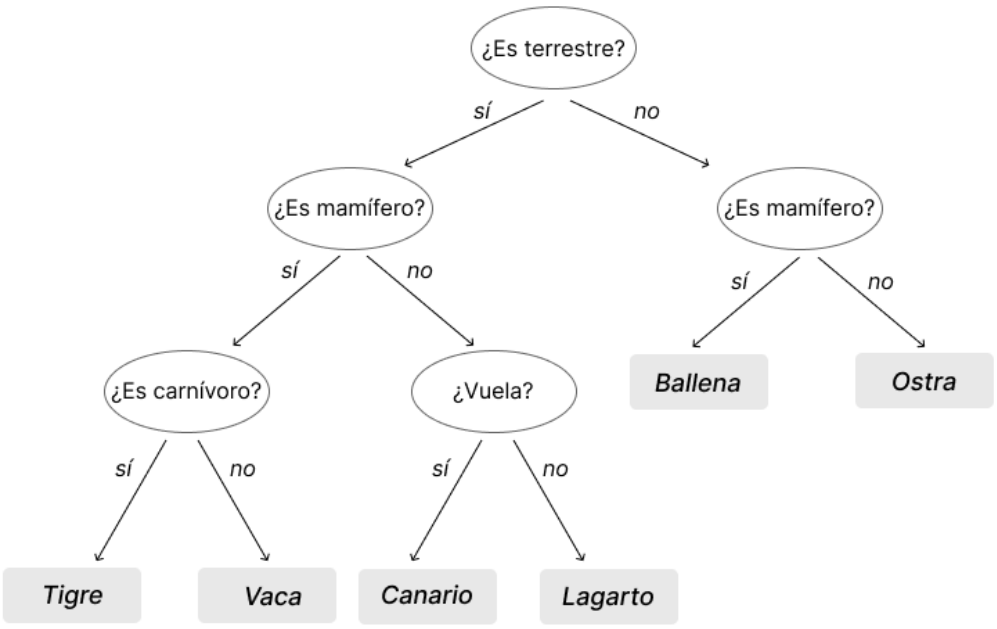

Figura 2.1: Ejemplo de árbol de decisión (muy simple) para clasificar animales.

Dicho de otra forma, en cada nodo se elige el atributo y sus valores que posteriormente permitirá tomar una decisión en la etapa de predicción. Cuando se inicia la construcción del árbol, todos los datos de ejemplo comienzan en el nodo raíz y la recursión finaliza cuando el subconjunto en un nodo tiene todo el mismo valor de la variable a predecir, o cuando dividir el nodo ya no agrega valor a las predicciones. Para dividir los datos en cada nodo, se suelen utilizar diferentes métricas para determinar el mejor atributo para cada caso. Un ejemplo de estas métricas es la *impureza de Gini* que a grandes rasgos evalúa cuán homogéneos son los valores de la variable a predecir dentro de un conjunto de datos.

La gran ventaja de los árboles de decisión es que permiten interpretar los resultados obtenidos. Es decir que a partir de un nuevo dato es posible "recorrer.[el] árbol para determinar la secuencia de decisiones que llevan a la predicción obtenida, lo que también permite identificar la importancia que cobra cada atributo de los datos en la toma de decisiones. Esta característica no la tienen los modelos de caja negra como la redes neuronales, en los que no es posible explicar con certeza cómo se llega a los resultados obtenidos.

Respecto a las limitaciones de este tipo de algoritmo, la más es importante es que los árboles generados pueden llegar a ser muy complejos, lo cual hace que sean propensos a

generar *sobre-ajuste*. El sobre-ajuste (también llamado sobre-entrenamiento) ocurre cuando un modelo está muy ajustado a los datos de ejemplo y no es capaz de predecir correctamente nuevos datos. Para evitar que esto suceda, se suelen utilizar distintos mecanismos para reducir el tamaño de los árboles generados.

2.9.2. Aprendizaje automático en UX

La popularidad que ha cobrado la inteligencia artificial en los últimos años dio lugar al desarrollo de nuevas herramientas que, haciendo uso de distintas técnicas de aprendizaje automático, permiten crear productos digitales con una mejor experiencia de usuario, dentro de ciclos de desarrollo más cortos y a un costo más bajo. Mediante la automatización de tareas que requieren intervención manual, estas herramientas tienen por objetivo dar soporte en las distintas etapas de un proceso de diseño centrado en el usuario.

Las etapas iniciales de un proceso de DCU tienen que ver con entender en el contexto en el cual se usará el producto a desarrollar y con especificar sus requerimientos. Allí es fundamental recolectar mucha información de posibles usuarios para identificar las necesidades que se pretenden resolver. En este sentido, Salminen et al. desarrollaron un algoritmo para automatizar la generación de personas y perfiles de usuario [Salminen et al., 2019]. Por otra parte, Suleri et al. propusieron el uso de redes neuronales profundas para convertir prototipos de baja fidelidad en alta, con el propósito de ayudar a que los diseñadores puedan elaborar sus ideas rápidamente [Suleri et al., 2019].

Una vez que se especifican los requerimientos, el siguiente paso es el diseño de la solución. Esto generalmente implica la creación de prototipos que se van modificando iterativamente hasta que se llega a un diseño definitivo. Por este motivo, una de las aplicaciones del aprendizaje automático en esta etapa es automatizar la evolución de los prototipos. Por ejemplo, Buschek et al. desarrollaron una herramienta que permite convertir prototipos hechos en papel en artefactos digitales [Buschek et al., 2020]. Otra aplicación es la optimización de diseños mediante la sugerencia de distintas alternativas que un diseñador puede seleccionar [Todi et al., 2016]. En este trabajo, Todi et al. proponen una herramienta de prototipado que ofrece opciones de diseño a medida que el diseñador avanza con el prototipo. La característica que tiene esta herramienta es que está destinada al diseño inicial de una UI, mientras que en esta tesis la generación de alternativas de diseño se realiza tomando como base una UI ya desarrollada.

Respecto a la evaluación de diseños, el uso de logs de interacción ya ha sido propuesto para evaluar la UX en aplicaciones móviles [Yang et al., 2020]. En ese trabajo, Yang et al. proponen capturar información de los clicks realizados que luego se utiliza como entrada de

una red neuronal profunda para simular de algún modo la experiencia de los usuarios. Si bien ese trabajo es similar a lo que se propone en esta tesis, tiene la particularidad de estar destinado únicamente a la evaluación de aplicaciones móviles, y además tiene como limitación que solo se basa en un único aspecto de la interacción de usuario; los clicks realizados. En el ámbito de las aplicaciones móviles también se puede mencionar el trabajo de Zhou et al. [Zhou et al., 2020], el cual propone un framework para comparar y evaluar alternativas de diseño en función de las preferencias de los usuarios, sin considerar la interacción. Dentro de las propuestas que evalúan aspectos subjetivos de los usuarios también se pueden incluir los trabajos ya mencionados de Dou et al. y Michailidou et al. que a partir de ratings asignados por los usuarios desarrollan modelos para predecir las propiedades estéticas de una página web [Dou et al., 2019, Michailidou et al., 2021]. Volviendo a los trabajos que utilizan aprendizaje automático con logs de interacción, probablemente el más relevante y cercano a esta propuesta es el de Speicher et al. [Speicher et al., 2014]. Otros trabajos como el de Li et al. y Leino et al. proponen predecir aspectos de la interacción de los usuarios pero sin utilizar datos de interacción [Li et al., 2018, Leino et al., 2019]. Ambos trabajos intentan estimar el tiempo requerido por un usuario para completar una tarea específica; el primero lo hace utilizando redes neuronales profundas y el segundo usa una variante de aprendizaje automático llamada *aprendizaje por refuerzo* para determinar la secuencia de operaciones que realiza un usuario en una UI, para luego poder calcular el tiempo de tarea a través del modelo KLM.

Finalmente, considerando que los diseños realizados tienen que implementarse, existen trabajos que intentan automatizar esta tarea mediante la transformación de prototipos en código [Beltramelli, 2018, Moran et al., 2020]. Si bien el aprendizaje automático tiene un gran potencial para automatizar la generación de código, aún queda mucho por hacer en cuanto a la validación de resultados con desarrolladores y diseñadores.

Capítulo 3

Un método para desarrollar y evaluar UX refactorings

Este capítulo proporciona un pantallazo general de la solución propuesta. Habiendo definido el marco teórico necesario y los trabajos relacionados, primero se describen algunas limitaciones actuales del uso de los CSWRs, y luego se describen los desarrollos realizados para cumplir con los 2 objetivos específicos detallados en el capítulo 1.

3.1. Solución propuesta

En el capítulo anterior se describió la importancia de la técnica de refactoring para garantizar la mantenibilidad del código fuente desarrollado y de cómo esta idea fue posteriormente utilizada para mejorar atributos de calidad externos de una aplicación como su usabilidad. Así como existen herramientas para aplicar automáticamente los refactorings de código, también se han propuesto herramientas para automatizar tanto la detección de usability smells como también la aplicación de refactorings, y así facilitar la evaluación y mejora de la UX. En particular, en la sección 2.3 se habló de Kobold, una herramienta que identifica los smells a partir del análisis de la interacción de los usuarios que luego pueden solucionarse aplicando automáticamente CSWRs. Si bien Kobold fue propuesto como un servicio para mejorar la usabilidad automáticamente, la herramienta posee una serie de limitaciones que motivaron la propuesta de este trabajo.

La primera limitación está relacionada con no poder determinar el impacto que tienen los refactorings en cada caso particular. En Kobold, una vez que se aplica un CSWR, los cambios generados son visibles para todos los usuarios de la aplicación destino, por lo cual se modifica

la interfaz de usuario sin antes evaluar estos cambios. Esto puede ser contraproducente ya que si bien los refactorings fueron diseñados para mejorar la experiencia de usuario [Grigera et al., 2016], puede ocurrir que en ciertos casos provoquen efectos negativos. La necesidad de evaluar los refactorings se vuelve más evidente en los usability smells que pueden ser resueltos por refactorings alternativos, como el caso del smell Unformatted Input que tiene 2 posibles soluciones (ver Figura 3.1). Elegir una de las 2 opciones puede ser complejo para el usuario de Kobold, principalmente porque no se puede determinar a priori qué refactoring funcionará mejor en cada situación específica.

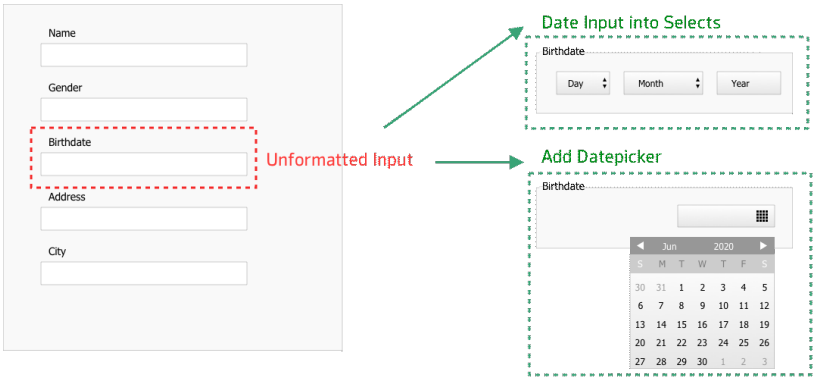

Figura 3.1: Refactorings alternativos para el usability smell *Unformatted Input*.

La otra limitación que tiene Kobold es que los refactorings se aplican únicamente en respuesta a los usability smells identificados. Esto quiere decir que para poder refactorizar la UI, primero es necesario recolectar una cantidad considerable de interacciones de usuario para detectar los usability smells correspondientes, y luego poder aplicar los refactorings asociados a éstos. Además, hay algunos refactorings que no se pueden aplicar porque la herramienta no puede inferir automáticamente ciertos parámetros. Por lo cual en esos casos los refactorings solo aparecen a modo de sugerencia y queda a criterio del usuario cómo implementarlos.

Teniendo en cuenta las limitaciones anteriores y la falta de artefactos mencionada en la sección previa, en este trabajo se propone un **método para explorar variantes de diseño en una aplicación utilizando la técnica de refactoring, y un mecanismo para evaluar y comparar estas variantes de diseño generadas**.

3.1.1. Generación de variantes de diseño

El método para explorar variantes de diseño consiste básicamente en el uso de los refactorings para visualizar y evaluar cambios en la UI. A diferencia de Kobold que utiliza los refactorings como soluciones persistentes a los problemas de usabilidad reportados, el objetivo de este método es que un diseñador de UX pueda aplicar refactorings libremente, ya sea para explorar un posible cambio en la UI, validar una idea de diseño, etc. Los refactorings se aplican de forma asistida a través de los CSWRs, que al ser scripts parametrizables que se inyectan en las páginas web, permiten refactorizar la UI sin la necesidad de codificar los cambios. Este aspecto resulta fundamental para los diseñadores de UX, que generalmente no cuentan con conocimientos de programación. De esta manera, un diseñador puede explorar cambios de diseño rápidamente y sin tener que depender de los desarrolladores para implementarlos.

El método fue implementado en una herramienta llamada UX-Painter; una extensión web que permite aplicar los CSWRs utilizando técnicas de End-user Development. Básicamente, cada refactoring se aplica eligiendo el elemento a refactorizar en la misma UI e ingresando ciertos parámetros adicionales. Los refactorings aplicados se pueden guardar en diferentes versiones de la aplicación, que posteriormente pueden ser evaluadas. La gran ventaja de esta herramienta es que los CSWRs no alteran la funcionalidad de la aplicación, por lo cual las versiones generadas se pueden evaluar en la misma UI real.

El hecho de utilizar los CSWRs como medio para explorar cambios de diseño, hizo que se desarrollen ciertas extensiones y mejoras a la técnica de refactoring que fueron plasmadas en UX-Painter:

- Se automatizaron los refactorings que no se podían aplicar en Kobold. Los refactorings como *Distribute Menu* y *Split Page* que antes eran sugeridos, en UX-Painter se pueden aplicar porque los parámetros de los refactorings no se infieren automáticamente, sino que el usuario los selecciona en la misma UI.

- Se incorporaron nuevos refactorings al catálogo con el objetivo de proveer más alternativas de diseño. Ejemplos de nuevos refactorings son *Format Input* y *Turn Select into Autocomplete*.

- Se incorporó a cada CSWR un mecanismo de adaptación de estilos. Teniendo en cuenta que los CSWRs son scripts externos que modifican el DOM de una página web, pueden ocasionar alteraciones no deseadas en los estilos de la página. Para minimizar esto se

desarrolló un mecanismo que permite adaptar el estilo de los elementos refactorizados al estilo de la página en cuestión.

- Se desarrolló un mecanismo para generar una implementación preliminar de los refactorings. Las versiones creadas en UX-Painter no son cambios persistentes, sino que son cambios volátiles que se almacenan en el navegador con el objetivo de facilitar la evaluación de alternativas de diseño. Una vez que se comprueba que una versión mejora la experiencia de usuario, sus refactorings deben implementarse en el código de la aplicación. Para facilitar esta tarea, el mecanismo desarrollado permite generar una implementación preliminar de los refactorings que los desarrolladores pueden utilizar como base para su propia implementación. En este sentido, UX-Painter no solo sirve para explorar cambios de diseño, sino que además facilita la comunicación entre el equipo de UX y los desarrolladores cuando estos cambios deben implementarse.

3.1.2. Evaluación y comparación de variantes de diseño

El propósito del método anterior es generar rápidamente alternativas de diseño para luego poder evaluarlas y finalmente implementar la que funciona mejor. Es decir que una vez que se generan las versiones con UX-Painter, la idea es poder determinar de algún modo el impacto de los refactorings en la aplicación destino, de manera de asegurar que los cambios de diseño que se implementen en ésta produzcan una mejora en la experiencia de usuario. No solamente es importante poder evaluar si un refactoring produce una mejora o no, sino que además también es necesario poder comparar alternativas como el ejemplo de la Figura 3.1 para determinar cuál es más apropiada en un contexto específico.

Para determinar el impacto de los refactorings se desarrolló una métrica denominada *esfuerzo de interacción*. Se trata de un puntaje asignado por expertos en UX a cada interacción de usuario con los diferentes elementos interactivos (widgets) que se ven modificados por los refactorings. Este puntaje va de 1 a 4; un mayor valor implica más esfuerzo por parte del usuario. Al ser un valor unificado para los diferentes tipos de widgets, permite comparar widgets alternativos que sirven para un mismo propósito como los de la Figura 3.1.

Teniendo en cuenta que el esfuerzo es asignado por expertos en UX en función de lo que observan en la interacción del usuario, se desarrollaron modelos de predicción para calcularlo automáticamente en 6 tipos de widget: campos de texto, links, menús desplegables, botones de radio, datepickers y menús desplegables de fecha. La entrada de los modelos es un conjunto de micro-medidas obtenidas de la misma interacción. Estas micro-medidas

evalúan aspectos tales como tiempos, movimiento de mouse, eventos de teclado, etc. y fueron desarrolladas específicamente para esta tarea mediante diferentes pruebas con expertos en UX y usuarios finales. Por otro lado, el entrenamiento de los modelos requirió de un proceso de recolección de datos en el que también participaron usuarios finales para generar las interacciones, y expertos en UX que asignaron el esfuerzo de interacción a cada una de éstas. Este proceso fue completamente remoto, por lo cual fue necesario el desarrollo de herramientas de soporte.

Finalmente, para visualizar el esfuerzo de interacción se desarrolló una aplicación web llamada UX-Analyzer. Esta herramienta recibe interacciones de usuario de la aplicación bajo análisis y utiliza los modelos de predicción para estimar el esfuerzo de cada una. En UX-Analyzer el usuario, que puede ser un miembro del equipo de UX o cualquier otro interesado, puede crear evaluaciones para monitorear el esfuerzo de interacción de las distintas páginas de su aplicación. A su vez dentro de cada evaluación se pueden generar diferentes versiones para analizar diseños alternativos. De cada versión no solamente se puede observar el esfuerzo de interacción de los usuarios en cada widget, sino que además se muestra el esfuerzo global de la versión que facilita la comparación de alternativas.

3.2. Proceso de mejora de la experiencia de usuario

La solución propuesta tiene por objetivo mejorar la experiencia de usuario de una UI, por lo cual para utilizar las herramientas propuestas es necesario contar con una aplicación web funcionando en un navegador. Esto no quiere decir que la UI bajo análisis tenga que estar en producción expuesta a muchos usuarios, ya que la solución también es aplicable a una UI que aún se encuentra en etapa de desarrollo. Un ejemplo de esto último podría ser el caso de un incremento de producto recientemente desarrollado y que debe ser sometido a pruebas antes de ser lanzado.

Las herramientas desarrolladas entran en acción cuando los expertos en UX identifican en una UI potenciales deficiencias de diseño o problemas de interacción que deben modificarse para mejorar la experiencia de usuario. Estos problemas de interacción pueden ser usability smells, que en el caso de una UI en producción son detectables automáticamente a través de Kobold, o bien pueden ser identificados manualmente por los expertos mediante pruebas con usuarios o de inspección. Más allá de las deficiencias de diseño y los usability smells, las herramientas también son útiles para simplemente validar ideas de diseño o explorar alternativas. La Figura 3.2 ilustra el proceso en el cual se utilizan estas herramientas.

Frente a la necesidad de realizar cambios en una UI, ya sea para solucionar problemas de

interacción o para explorar variantes, los expertos en UX pueden utilizar UX-Painter para aplicar estas modificaciones mediante los CSWRs y visualizar los resultados en el navegador (ver Figura 3.2/a). La posibilidad de hacer cambios rápidamente y sin tener que codificarlos hace que esta herramienta sea especialmente útil en métodos de desarrollo ágiles donde los tiempos son muy acotados.

Luego de generar versiones alternativas de una aplicación, llega el momento de evaluarlas para determinar el impacto que tiene cada una en la experiencia de los usuarios. Esto puede apreciarse en la Figura 3.2/b. En este punto es donde se puede utilizar UX-Analyzer para comparar las versiones generadas en términos del esfuerzo de interacción que resulta de cada una. Esto implicaría que los expertos en UX realicen pruebas con usuarios en cada versión bajo análisis en las que se utilice el script provisto por UX-Analyzer. Este script es el encargado de calcular las micro-medidas de cada interacción de usuario, y posteriormente enviarlas al servidor de la herramienta para obtener el esfuerzo de interacción de cada versión. El esfuerzo resultante de las versiones le permite a los expertos en UX determinar cuál de estas versiones se desempeña mejor.

Finalmente, una vez que los expertos en UX determinen la mejor versión a través de UX-Analyzer, los desarrolladores del equipo deben implementar en la aplicación bajo análisis los CSWRs de la versión seleccionada (Figura 3.2/c). Allí es donde los desarrolladores pueden hacer uso de UX-Painter para visualizar una implementación preliminar de los CSWRs seleccionados, que puede servirles para ahorrar tiempo en las tareas de codificación.

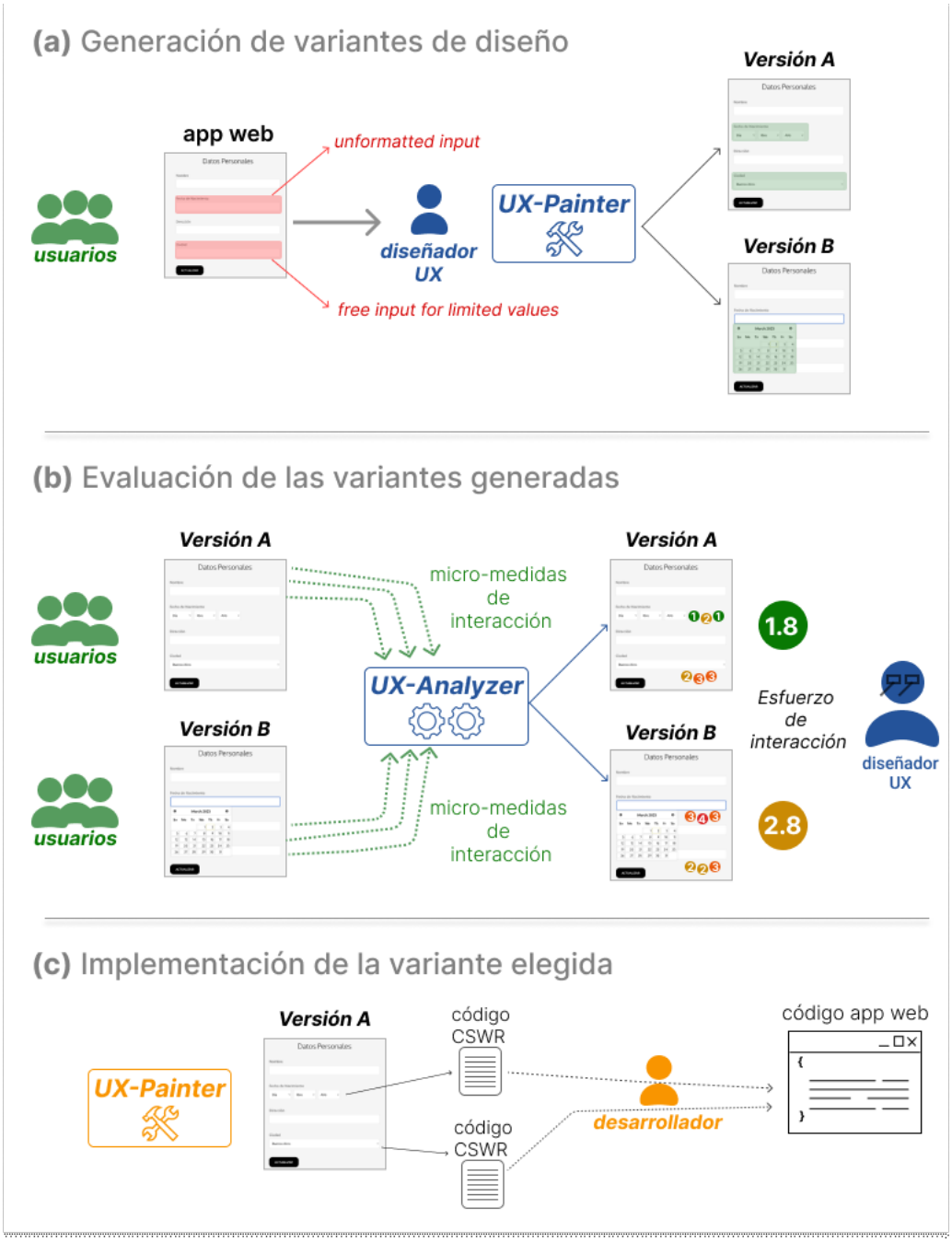

Figura 3.2: Proceso de mejora de la UX asistido por las herramientas desarrolladas.

Capítulo 4

Variantes de diseño de UI a través de refactorings

Este capítulo describe en detalle la solución propuesta para el primer objetivo específico; el método para generar variantes de diseño a partir de los *Client-side web refactorings*. Luego de hacer un breve repaso de la motivación y de presentar los fundamentos de esta propuesta, se describe el catálogo de CSWRs (incluyendo los nuevos) y el mecanismo de adaptación de estilos incorporado. Posteriormente, se muestra el funcionamiento de UX-Painter con algunos ejemplos, y para finalizar se ubica esta herramienta en el contexto de un método de desarrollo ágil.

4.1. Refactorings para explorar variantes de diseño

En trabajos previos, la técnica de UX refactoring fue propuesta con el objetivo de proveer asistencia automática tanto en la detección de problemas de UX como en la aplicación de soluciones [Grigera et al., 2017a]. Esto fue pensado para equipos de desarrollo que no cuentan con la posibilidad de realizar otro tipo de evaluaciones de UX, ya sea por falta de recursos o limitaciones de tiempo. En el capítulo 3 se adelantaron las limitaciones de la herramienta Kobold, que tienen que ver con: (1) aplicar los CSWRs directamente en las páginas web que se encuentran en producción, haciendo que los cambios sean visibles a todos los usuarios, y (2) los CSWRs sólo pueden utilizarse luego de identificarse la presencia de usability smells.

En un principio se resolvió la primera limitación extendiendo Kobold para dar la posibilidad de visualizar los CSWRs en diferentes versiones de una aplicación [Grigera et al., 2018].

A partir de esta modificación, los usuarios de la herramienta pueden registrar diferentes versiones de la aplicación, que luego se pueden seleccionar como destino al momento de aplicar un CSWRs. De esta manera, por ejemplo ante la presencia de un smell con soluciones alternativas como *Unformatted Input*, es posible aplicar cada CSWRs en una versión diferente. Para visualizar cada versión, los usuarios de la aplicación deben utilizar un parámetro en la URL cuyo valor identifica la versión y genera que el script de Kobold descargue y ejecute únicamente los CSWRs aplicados en esa versión.

Si bien la posibilidad de versionar una aplicación en Kobold permite evaluar refactorings alternativos incluso sin que éstos sean visibles para todos los usuarios, esta mejora no resuelve la limitación (2) porque el proceso de aplicar un CSWRs sigue siendo el mismo, con el único agregado de tener que elegir una versión destino. Por este motivo, se decidió cambiar el enfoque de los CSWRs y utilizarlos como medio para configurar y explorar diseños de UI alternativos libremente, es decir, sin requerir de la presencia de un usability smell. La idea es que los diseñadores de UX, que son los encargados de crear la UI, puedan usar los CSWRs para inspeccionar y evaluar diseños alternativos. Teniendo en cuenta que los CSWRs son scripts predefinidos que pueden ser parametrizables, se implementó un método que permite aplicarlos de manera asistida sin tener que codificarlos. Esto tiene dos objetivos principales: (1) aligerar los cambios de diseño, lo cual es fundamental para los tiempos acotados del desarrollo ágil, y (2) poder realizar cambios sin necesitar conocimientos de programación para que los diseñadores de UX no dependan de los desarrolladores.

4.2. UX-Painter

4.2.1. Contexto

Como se adelantó en el Capítulo 2, si bien se reconoce la importancia de integrar la gestión de la UX dentro de las métodos ágiles, aún es difícil sincronizar las prácticas de los equipos de UX con las de los desarrolladores. La mayoría de los artefactos que utilizan los desarrolladores y los diseñadores de UX para comunicarse están destinados a etapas previas al desarrollo, como la elicitación de requerimientos y el diseño inicial de la UI. Por lo cual, existe una falta de soporte para mejorar de una interfaz ya desarrollada, es decir, para detectar problemas de UX y explorar/evaluar posibles soluciones a éstos.

Si bien existen herramientas que ayudan a los equipos de UX a identificar problemas a partir del feedback de los usuarios, como las que realizan logging de interacción o las que permiten hacer user testing remoto, en general este tipo de herramientas no proveen

soporte para solucionar los problemas que reportan. Para explorar cambios en la UI, se suelen utilizar prototipos que en cierta medida simulan la interacción con la aplicación real. En este sentido, trabajar sobre prototipos sirve para identificar problemas en forma temprana, pero por otro lado, dado que se trata de una simulación, es posible que no se reflejen ciertos problemas que experimentan los usuarios finales en la aplicación. Por este motivo, es importante también realizar evaluaciones en el contexto real de la aplicación. Sin embargo, esto muchas veces resulta imposible debido al alto costo que requiere implementar todos los cambios de diseño a explorar. Además de ser una tarea que consume mucho tiempo, es necesario contar con conocimientos de programación, por lo cual es probable que el equipo de UX no lo pueda hacer por su cuenta. Esto genera que los problemas de UX se acumulen y que se apliquen cambios de diseño sin evaluarse, lo cual puede tener un impacto negativo en la experiencia de los usuarios en lugar de producir una mejora.

Para mitigar las limitaciones detalladas anteriormente, el objetivo del método propuesto es facilitar la evaluación en la UI real de posibles soluciones a problemas de UX previamente identificados, o incluso observar diferentes ideas de diseño rápidamente. Tal como se mencionó en el Capítulo 2, si bien la exploración de alternativas es una práctica común que se realiza durante el diseño de la UI utilizando diferentes herramientas de prototipado, en este caso la idea es poder llevar adelante esta práctica en una etapa posterior, cuando la UI ya fue desarrollada y se encuentra en producción.

Este método de exploración usando los CSWRs se desarrolló en una herramienta denominada UX-Painter [Gardey et al., 2020]. Se trata de una herramienta de *End-User Development* [Ko et al., 2011] que permite a los diseñadores de UX realizar cambios en la UI utilizando técnicas de programación visual.

4.2.2. Requerimientos

Teniendo en cuenta que el objetivo de UX-Painter es dar soporte a la evaluación de la UX de una interfaz de usuario ya desarrollada, se plantearon los siguientes requerimientos a ser cumplidos por la herramienta:

(a) Permitir explorar variantes de diseño en la interfaz de usuario real. Dado que ya se cuenta con una UI desplegada (ya sea en un entorno de testing o de producción), el objetivo es poder evaluar cambios de diseño en entornos de prueba de la misma aplicación con usuarios reales.

(b) Poder modificar la UI de cualquier aplicación web sin importar la tecnología con la que es desarrollada.

(c) Poder crear variantes de diseño sin tener que codificarlas. Como ya se mencionó antes, la principal limitación de realizar pruebas en la aplicación real es la necesidad de tener que implementar los cambios a evaluar. Poder aplicar modificaciones a una UI sin tener que escribir código no solo ahorra un tiempo considerable, sino que además permite que las haga una persona sin conocimientos de programación. Es por esto que se decidió adoptar una estrategia de EUD para dar soporte a los equipos de UX.

(d) Crear cambios de diseño rápidamente. Esto está relacionado con el punto anterior. No solo se refiere al tiempo necesario para crear nuevos diseños, sino también a la flexibilidad de poder observar e interactuar con estos, pudiendo descartarlos fácilmente sin afectar la funcionalidad de la aplicación.

(e) Persistir los diseños creados. Es necesario contar con un mecanismo de persistencia que permita guardar los cambios aplicados para luego poder evaluarlos (por ejemplo a través de A/B testing o pruebas con usuarios) y finalmente comunicarle al equipo de desarrollo las modificaciones que deben ser implementadas.

(f) Re-crear las variantes de diseño generadas en otros entornos, para permitir la evaluación de éstos en el contexto de los usuarios.

(g) Llevar un registro de los cambios que se realizan en la UI con el objetivo de poder comunicárselos al equipo de desarrollo cuando deben ser implementados. Esto apunta a reducir los problemas de comunicación, dado que teniendo un registro de las modificaciones aplicadas junto con la UI resultante, los desarrolladores pueden comprender mejor qué tienen que codificar.

Generar los cambios de diseño en el navegador es lo que permite satisfacer los tres primeros requerimientos antes descriptos. El navegador ha sido ampliamente usado como medio para modificar la UI externamente, por ejemplo en herramientas de aumentación web para permitirle a usuarios finales personalizar aplicaciones [Díaz et al., 2016]. En este caso, quienes producen cambios en la UI son los diseñadores de UX, y los usuarios finales interactúan con esos cambios de diseño en las pruebas que se realicen.

En cuanto a los siguientes dos requerimientos ((d) y (e)), aplicando los CSWRs a través de una estrategia de EUD, es posible crear variantes de diseño rápidamente y sin codificarlas. Dado que los refactorings no alteran la funcionalidad, los usuarios pueden continuar usando la aplicación en cualquiera de las variantes. Los refactorings también sirven para comunicar los cambios aplicados ((g)), porque cada refactoring es una transformación muy específica sobre un conjunto de elementos o partes de una UI, que tiene una semántica bien definida.

Por último, para poder persistir los diseños, se utiliza el concepto de *versiones de la aplicación* para representar una secuencia de refactorings que tiene por objetivo lograr cambios de diseño más grandes. Estas versiones se guardan en el almacenamiento local del navegador.

4.2.3. Catálogo de refactorings

En este trabajo se ha extendido el catálogo de CSWRs presentados en trabajos previos [Grigera et al., 2017a] principalmente para proveer distintas alternativas para solucionar un mismo problema de interacción. Contar con más de una solución posible en términos de CSWRs, permite a los diseñadores de UX crear versiones alternativas de una aplicación web para evaluar cuál de éstas funciona mejor. Si bien cada CSWR realiza cambios pequeños, se pueden aplicar en secuencia para lograr cambios de diseño mayores. Todos los CSWRs preservan la funcionalidad de la aplicación reemplazando widgets de la UI con otros semánticamente similares, o simplemente re-ordenando elementos que ya existen en la misma.

A continuación se describen los 19 CSWRs incluidos en UX-Painter, junto con los parámetros necesarios para aplicarlos y el problema de UX que cada CSWR intenta resolver:

Add Autocomplete

Incorpora auto-completado en un campo de texto para sugerir una lista de valores esperados mientras el usuario tipea, con el objetivo de facilitar el ingreso de valores populares. Al momento de aplicarlo, el usuario debe ingresar la lista de valores a sugerir.

Add Datepicker

Agrega un calendario a un campo de texto para seleccionar una fecha. Éste puede ser útil para prevenir errores de formato que suelen ocurrir cuando se ingresa una fecha manualmente. A diferencia del refactoring anterior, no necesita ningún parámetro por parte del usuario.

Add Late Form Validation

Incorpora validación en el cliente a un conjunto de campos obligatorios de un formulario, que se realiza cuando el usuario envía el mismo. Este tipo de validación intenta minimizar

la cantidad de envíos fallidos del formulario. Para aplicar este CSWR, el usuario debe seleccionar la lista de los campos que son obligatorios.

Add Inline Form Validation

Es similar al CSWR anterior con la única diferencia que la validación de cada campo se realiza cuando el usuario abandona el mismo, es decir, cuando se pierde el foco. Validar los campos a medida que el usuario interactúa con cada uno, en ciertos casos puede ser más conveniente que validar todos los campos juntos al finalizar el formulario, para no abrumar al usuario con mensajes de error.

Add Link

Crea un enlace en una sección específica de una página con la finalidad de crear un atajo de navegación, evitando que el usuario tenga que atravesar diferentes páginas intermedias para llegar a la página destino. Para agregar el enlace, el usuario debe seleccionar la sección en la página e ingresar el nombre y la URL destino del enlace.

Add Loading Overlay

Agrega un overlay (un panel transparente que oscurece la pantalla) cuando se envía un formulario, para indicarle al usuario que la aplicación está procesando información. El feedback brindado por el overlay ayuda a prevenir la confusión del usuario que se suele generar cuando la UI no responde producto de una operación que toma tiempo.

Add Tooltip

Añade una etiqueta emergente que se muestra cuando el usuario ubica el cursor sobre un elemento con el objeto de proveer una descripción adicional de éste. El elemento puede ser un texto, una imagen, un botón o un input. Requiere el texto a mostrar en el tooltip.

Date Input into Selects

Este CSWR reemplaza un campo de texto con tres campos de selección para elegir día, mes y año de una fecha. Es una alternativa a *Add Datepicker* dado que también intenta facilitar el ingreso de una fecha.

Distribute Menu

Dentro de una lista de ítems seleccionables con acciones en lote, permite agregar o "distribuirüna acción a cada elemento de la lista. Muchas veces las listas seleccionables están diseñadas para poder aplicar acciones sobre múltiples ítems a la vez, pero hacerlo para uno solo termina siendo tedioso. Es por eso que se añade un botón a cada ítem de la lista para que sea más fácil ejecutar la acción correspondiente. Este CSWR es más complejo que lo otros porque afecta a múltiples elementos de una página, entre los cuales se encuentran: la lista de ítems, el widget que permite seleccionar cada ítem (que podría ser el mismo ítem o un checkbox por ejemplo) y la acción en lote que debe ser replicada en cada ítem. Todos estos elementos los debe seleccionar el usuario al momento de aplicar este refactoring.

Es importante mencionar que, si bien este refactoring fue propuesto en el catálogo previo [Grigera et al., 2017a], no fue implementado, Kobold solo lo presenta a modo de sugerencia. Esto se debe a que no se puede aplicar automáticamente porque no es posible inferir todos sus parámetros mencionados anteriormente. En cambio en UX-Painter, como es el mismo usuario el que selecciona o ingresa los parámetros, sí es aplicable y por eso se agregó su implementación.

Format Input

Es uno de los refactorings nuevos que se agregaron al catálogo. Consiste en permitir que únicamente se ingrese un cierto conjunto de caracteres en un campo de texto. Al igual que *Add Datepicker* y *Date Input into Selects*, el objetivo de este CSWR es prevenir errores en el ingreso de valores que tienen un formato específico. La diferencia con los refactorings anteriores es que éste no solo sirve para el ingreso de fechas, sino que también es útil para otros valores con formato como por ejemplo números de tarjeta de crédito, de teléfono, entre otros. Este refactoring se implementa usando una librería externa que permite definir un conjunto de caracteres válidos para un campo a partir de lo que técnicamente se conoce como *máscara*, que podría decirse que es como una expresión regular pero mucho más simple.

Al momento de aplicar el refactoring, el usuario debe ingresar la máscara en función del formato que requiera el campo en cuestión. Un "0.ªdmite un dígito ([0-9]), una "Aüna letra ([a-zA-z]) y una "Sün dígito o una letra. Cualquier otro caracter es un valor literal que tendrá que ser ingresado por el usuario. Por ejemplo, si se quiere permitir el ingreso de una fecha en formato DD/MM/AAAA, la máscara a utilizar es "00/00/0000".

Link to Top

Cuando el usuario hace scroll hacia abajo, muestra un enlace en la parte inferior derecha de la página que permite volver directamente al principio de la misma. Esta opción es de gran utilidad en páginas con mucho contenido, donde el usuario tiene que hacer scroll para encontrar lo que está buscando. Para aplicar este CSWR no se requiere ningún parámetro adicional.

Rename Element

Cambia la etiqueta (label) de un elemento con el fin de hacerlo más descriptivo, al igual que *Add Tooltip*. Los elementos a renombrar pueden ser texto estático (un título, un párrafo, la etiqueta de un campo de formulario) y elementos interactivos como botones y enlaces. Al momento de ejecutar este CSWR, el usuario debe ingresar el nuevo nombre del elemento a modificar.

Resize Input

Modifica el ancho (width) de un campo de texto. El objetivo de este refactoring es ajustar el tamaño del campo a la longitud de los valores esperados. De esta manera, los usuario pueden tener una mejor idea del tipo de contenido que se espera.

Split Page

Divide el contenido de una página en diferentes secciones, permitiendo visualizar una sección a la vez mediante un menú que se agrega en una parte específica de la página. Este CSWR se utiliza para facilitar el acceso al contenido de páginas que se encuentran muy sobrecargadas.

Al igual que *Distribute Menu*, *Split Page* en Kobold es únicamente sugerido porque su aplicación no es complemente automatizable. Esto se debe a que no es posible interpretar el contenido de una página para crear secciones que tengan sentido. En UX-Painter, se implementó el refactoring haciendo que el mismo usuario defina las diferentes secciones en las cuales se dividirá la página, ingresando para cada una un nombre y seleccionando en la página los elementos que la componen. Luego de definir las secciones, el usuario también debe seleccionar la ubicación del menú que permitirá visualizar cada una de estas.

Turn Attribute into Link

Convierte un elemento estático de una página como un texto o una imagen en un link a una URL específica. Muchas veces los usuarios hacen click en ciertos elementos esperando una respuesta de la aplicación pero no obtienen ningún resultado. Un ejemplo de esto podría ser el logo de la aplicación, a través del cual el usuario espera poder navegar a la página principal de la misma. Este refactoring puede ayudar a mejorar la navegabilidad del sistema.

Turn Input into Radios

Reemplaza un campo de texto libre, en el cual se espera un conjunto limitado de valores, por un conjunto de botones de radio para poder seleccionar uno de estos valores en lugar de tener que ingresarlos manualmente. Para no alterar la funcionalidad de la aplicación, se agrega una opción "Otro"que muestra un campo de texto para permitirle al usuario ingresar un valor no contemplado en las opciones predefinidas. La aplicación de este CSWR requiere que el usuario ingrese la lista de valores que se mostrarán en los botones de radio.

Al igual que *Add Autocomplete*, *Turn Input into Radios* pretende simplificar el ingreso de valores recurrentes. La diferencia está en que este último probablemente sea más apropiado cuando las opciones son pocas, dado que al mostrarse todas al mismo tiempo en pantalla, una larga lista de opciones puede dificultarle al usuario buscar y encontrar la opción deseada.

Turn Input into Select

Similar al caso anterior, sustituye un campo de texto con un menú desplegable con una lista de opciones predefinidas. La diferencia con *Turn Input into Radios* es que admite un mayor número de opciones, ya que la lista de opciones se muestra al desplegar el menú y además su contenido es *scrollable* para no ocupar toda la pantalla.

Este refactoring no formaba parte del catálogo previamente, fue agregado en UX-Painter. En cuanto a su implementación, es muy parecida a la de *Turn Input into Radios*; el campo de texto no se elimina sino que se oculta, y cada vez que el usuario elige una opción en el menú desplegable, se asigna el valor de la misma al campo de texto oculto para preservar la funcionalidad de la aplicación. Para más detalles de implementación, ver [Grigera, 2017], capítulo 6.

Turn Input into Textarea

Reemplaza un campo de texto convencional por uno que permite ingresar múltiples líneas de texto, formalmente conocido como *textarea*. Se trata de un nuevo CSWR agregado

al catálogo que surge como una alternativa a *Resize Input*, apropiado para indicarle al usuario que se espera que ingrese un texto largo.

Su implementación utiliza la misma estrategia que el caso anterior para reemplazar el widget; el campo original se oculta y lo que el usuario ingresa en el textarea agregado se guarda automáticamente en el campo oculto.

Turn Select into Autocomplete

Este CSWR podría considerarse el caso inverso a *Turn Input into Select* ya que cambia un menú desplegable con opciones predefinidas por un campo de texto libre que tiene autocompletado. La diferencia con *Add Autocomplete* es que los valores sugeridos no los ingresa el usuario, sino que son las mismas opciones contenidas en el menú desplegable.

También se trata de un nuevo refactoring que surgió observando menús desplegables con una gran cantidad de opciones que requerían de un tiempo considerable para encontrar la opción indicada, y en muchos casos este tiempo era mayor a ingresar el valor manualmente. Esta transformación permite que el usuario escriba parte del valor deseado y lo busque en una lista de opciones reducida.

4.2.4. Adaptación de estilos CSS

Como se describió al principio de este capítulo, los CSWRs realizan cambios en la UI modificando el DOM de la página subyacente. Estas modificaciones, que en general implican agregar o quitar ciertos elementos o widgets, pueden generar alteraciones no deseadas en la disposición de los elementos dentro de la página, formalmente conocida como *layout*. La Figura 4.1 muestra un ejemplo de una alteración del layout no deseada luego de aplicar el refactoring *Date Input into Selects* en el campo "ida". El problema allí es que los campos día, mes y año agregados por el refactoring, en lugar de mostrarse todos en la misma línea, se visualizan uno debajo de otro, lo que genera que el resto de los elementos de la sección se desplacen hacia abajo.

Por otro lado, más allá de las alteraciones en el layout que se puedan generar, también sucede que muchas veces los estilos CSS (colores, fuentes, etc.) de los nuevos elementos incorporados a la UI contrastan con los de la página destino, lo cual puede generar un impacto visual negativo en los usuarios. La Figura 4.2 ilustra la aplicación de *Turn Input into Select* en el campo "Ciudad". Allí se puede observar que el nuevo menú desplegable se ve muy diferente al que ya existe en la página ("País").

Con el objetivo de minimizar las alteraciones en el layout y el contraste visual en ciertas

Figura 4.1: El refactoring Date Input into Selects en el campo "ida"produce una alteración no deseada en el layout de la página.

propiedades estéticas, en este trabajo se extendieron los CSWRs incorporando un mecanismo de adaptación de estilos que genera determinadas propiedades CSS para que cada elemento modificado o agregado por los refactorings se ajuste a las propiedades visuales de la página destino. Este mecanismo básicamente consiste en extraer automáticamente propiedades CSS de la página y asignárselas a los elementos modificados por los CSWRs. La idea no es detectar y aplicar los estilos sin la intervención del usuario, sino que se trata de generar diferentes opciones posibles de estilos para los elementos refactorizados, de manera que el usuario de UX-Painter pueda inspeccionarlas y decidir cual de éstas aplicar.

Para extraer los estilos CSS de la página, se desarrolló un algoritmo que busca determinados tipos de widgets y extrae de estos un conjunto de valores para las propiedades CSS especificadas. Tanto el tipo de widget a buscar como las propiedades a extraer dependen del refactoring que se esté aplicando. Por ejemplo, en el caso del CSWR *Add Link* el algoritmo busca los colores y fuentes utilizados en otros links existentes, mientras que en *Turn Input into Selects* se extraen los bordes, colores y dimensiones de otros menús desplegables (si es que existen).

El algoritmo es configurable con diferentes estrategias de búsqueda que permiten priorizar los valores CSS detectados en la página en base a diferentes criterios. En este trabajo se definieron 2 estrategias de búsqueda que se utilizan en el proceso de adaptación de estilos de cada refactoring.

Búsqueda por Distancia

Esta estrategia busca los widgets de un determinado determinado tipo, y luego de recuperar los valores para el conjunto de propiedades CSS indicado, los ordena en base a la distancia euclidiana desde el widget asociado hacia el elemento objetivo. El ordenamiento se realiza de forma ascendente, de manera que se priorizan los estilos de los widgets cercanos al

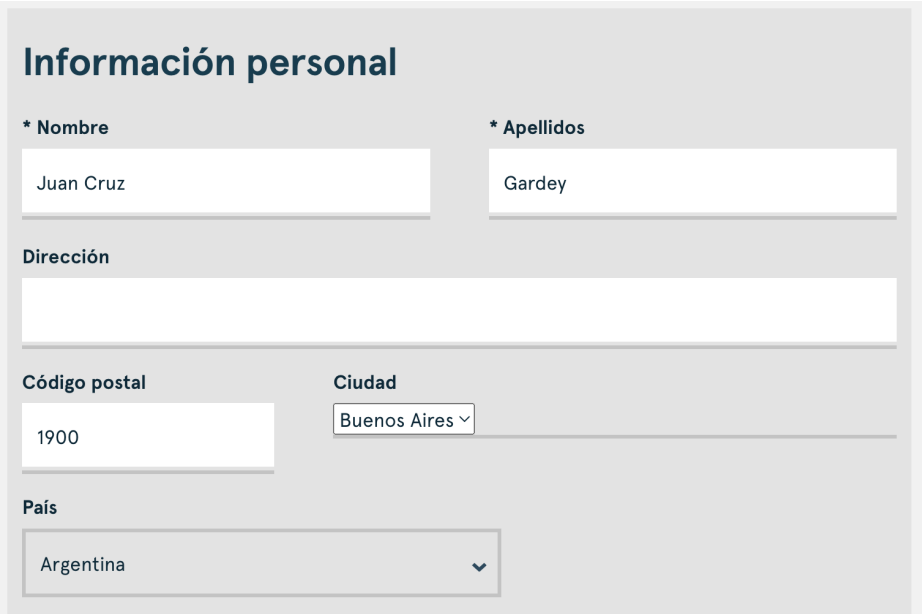

Figura 4.2: El estilo del nuevo menú desplegable "Ciudad" no coincide con el del menú desplegable de "País".

elemento objetivo por sobre aquellos cuyos widgets se encuentran más alejados visualmente. La idea detrás de esta estrategia es que muchas veces los estilos apropiados para un nuevo widget se encuentran dentro de la misma sección en la que éste esta ubicado. Por ejemplo, si se agrega un link en un menú, es muy probable que ese link se tenga que ver igual al resto de los links que ya existen allí. La Figura 4.3 ilustra esta situación, en la que luego de incorporar un link (*Nuevo Link*), el algoritmo identifica los diferentes tipos de link que existen, y prioriza los estilos del link más próximo (*Empretec*), dejando como segunda y tercer opción los links más alejados (*Arrepentimiento/Baja* y *Sucursales y Cajeros*).

Esta estrategia en general se utiliza cuando el nuevo widget no tiene un estilo predefinido, y los widgets existentes del mismo tipo tienen estilos que varían en función de la ubicación en la página. Tal es el caso de los links, campos de texto, menús desplegables y botones de radio.

Búsqueda por peso

Obtiene todos los valores de un conjunto de propiedades CSS que corresponden a un tipo de widget especificado, y los ordena de acuerdo a la cantidad de ocurrencias que tiene cada conjunto de valores. Se denomina peso (*weight*) a la cantidad de veces que se repite

Figura 4.3: Opciones de estilo para *Nuevo Link*. La primer opción es la del link más próximo, *Empretec*. Los estilos de *Arrepentimiento Baja* y *Sucursales y Cajeros* son la segunda y tercera opción, respectivamente.

un conjunto de valores. Esta estrategia de búsqueda permite recuperar los valores más usados para un conjunto de propiedades a lo largo de toda la página, por lo que se utiliza principalmente para ajustar ciertos estilos globales como colores y tipos de fuente. La Figura 4.4 muestra un ejemplo del refactoring *Add Datepicker* en el que se usa esta estrategia para obtener las combinaciones de colores más usadas en la página, que luego se aplican en el Datepicker incorporado.

Por último, es importante mencionar que si bien el algoritmo fue desarrollado para mejorar los CSWRs, también podría ser utilizado en otros casos en los que se requiera detectar y extraer automáticamente ciertas propiedades CSS de los widgets que existen en una página. En cuanto a las estrategias de búsqueda, el algoritmo puede extenderse fácilmente con nuevas variantes.

4.2.5. Arquitectura

UX-Painter es una extensión web que funciona completamente en el navegador. Esto permite aplicar CSWRs en cualquier página web y guardar los cambios como versiones alternativas de la aplicación destino. Las versiones se almacenan en el *local storage* del navegador, por lo cual la herramienta no depende de ningún servicio externo para usarse. Es suficiente con cargar la extensión en el navegador, sin la necesidad de realizar ninguna configuración adicional en la aplicación destino.

Cuando se crea una nueva versión, UX-Painter persiste para cada refactoring, todos los elementos necesarios para volver a ejecutarlo: tipo, elemento/s destino, parámetros ingresados por el usuario, estilos, y la URL de la aplicación en la que se ejecutó. Además de guardar el listado de versiones creadas en una aplicación, también se almacena la versión

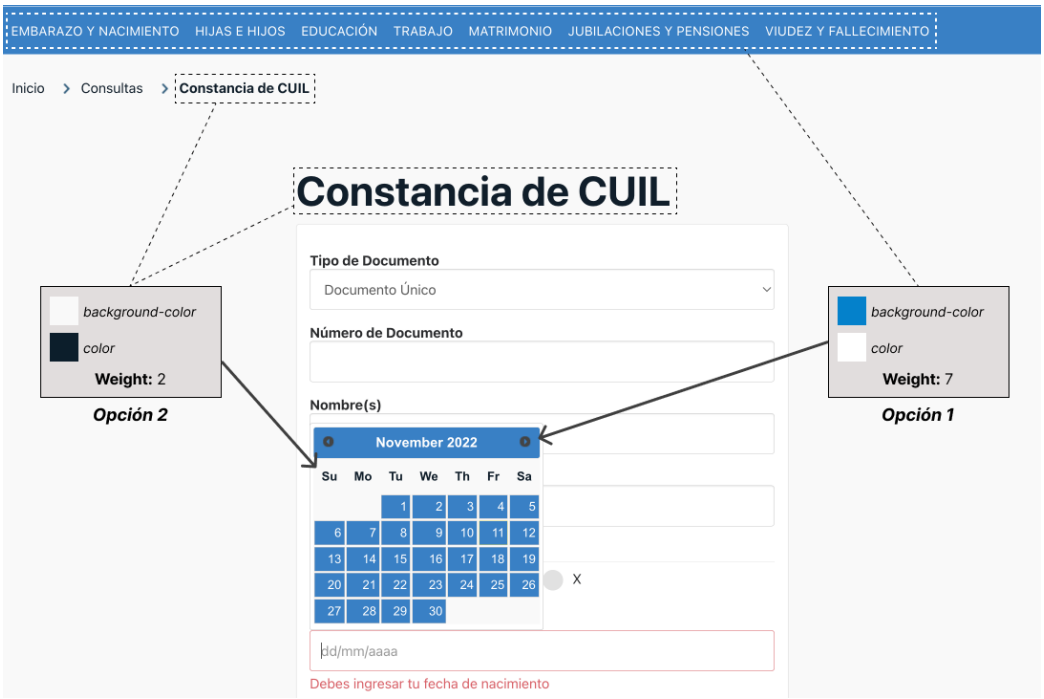

Figura 4.4: Opciones de estilo para el nuevo Datepicker. La primer opción que es la que se ve en el título y en los días del Datepicker. Ésta tiene peso 7 porque los 7 links de la barra de navegación utilizan estos colores. Por otro lado, la segunda opción posee un peso de 2 porque hay 2 elementos que tienen esa combinación de colores.

activa, es decir, la versión que se está mostrando en un momento dado. Cuando se carga una página, la herramienta automáticamente busca y ejecuta todos los refactorings de la versión activa que correspondan con la URL cargada.

El alcance de las versiones es una aplicación web completa, la cual se identifica por su nombre de dominio. Esto quiere decir que dentro de una aplicación específica, una misma versión puede contener CSWRs en diferentes páginas, permitiendo así lograr cambios de diseño más grandes.

4.2.6. Funcionamiento

A continuación se describe cómo funciona la herramienta aplicando algunos CSWRs en el formulario de checkout de Amazon.[1]

Lo primero que se ve al abrir la extensión web es el listado de versiones de la aplicación

[1] https://amazon.com

que está mostrando el navegador (ver Figura 4.5). Por defecto, solo existe la versión original, que no puede ser modificada para que el usuario siempre tenga la posibilidad de volver al diseño original. El primer paso para aplicar CSWRs es crear una nueva versión, para lo cual se requiere un nombre. Creada la nueva versión, se pueden aplicar refactorings en ésta. La Figura 4.6 ilustra el listado completo de CSWRs que se observa cuando el usuario selecciona la opción "Add Refactoring"dentro de una versión. El nombre de cada refactoring se muestra junto una opción que permite ver más detalles del mismo, como una breve descripción y dos animaciones que muestran la UI antes y después de aplicar el CSWRs (ver Figura 4.7).

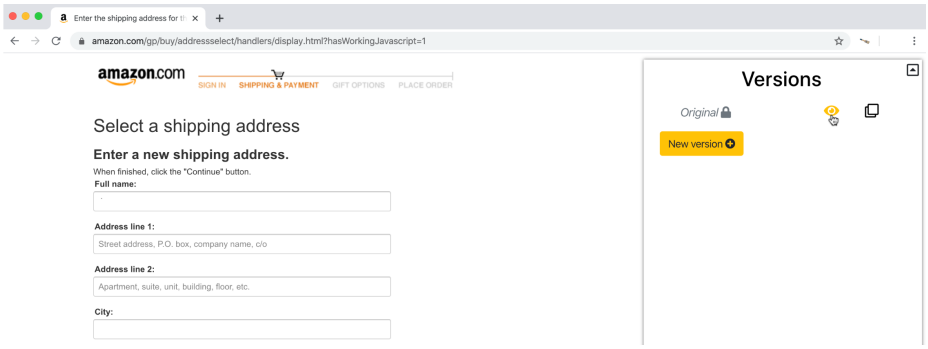

Figura 4.5: Listado de versiones de la aplicación actual. La primera vez que se abre la herramienta solo se ve la versión original que no puede modificarse.

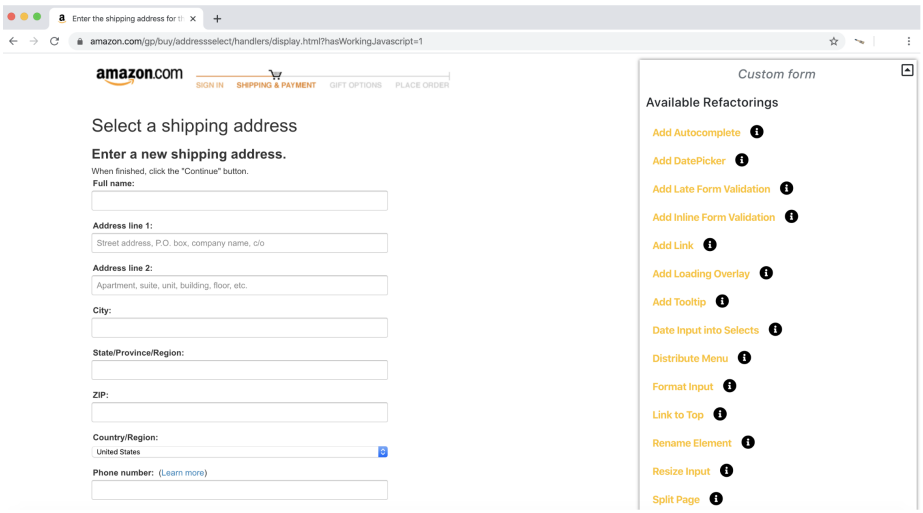

Figura 4.6: Listado completo de CSWRs. Cada refactoring tiene una opción de más información para que el usuario pueda entender mejor qué modificaciones realiza.

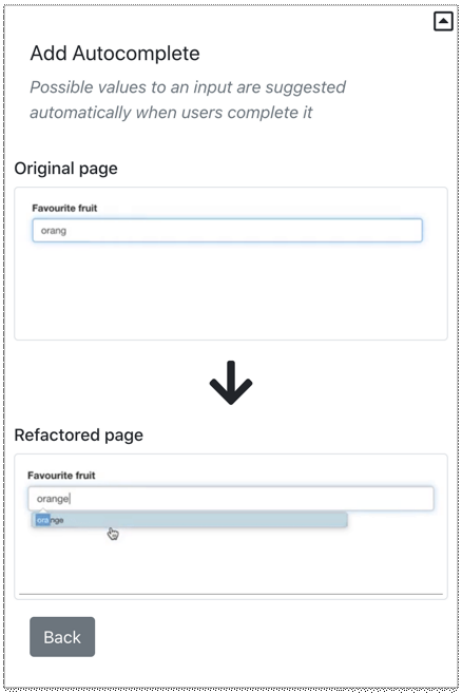

Figura 4.7: Descripción de Add Autocomplete. Se provee una breve descripción junto con un ejemplo del refactoring

Una mejora que se puede hacer en el formulario de checkout de Amazon es agregar validación en el cliente para minimizar la cantidad de envíos fallidos que suelen ocurrir por no completarse todos los campos. Como se describió en la sección 4.2.3, se pueden aplicar 2 tipos de validaciones: *inline* o *late*. En este ejemplo se aplicará *Add Inline Form Validation* pero el proceso de aplicación de ambos refactorings es prácticamente el mismo. Una vez que se elige el CSWR deseado en el listado, la herramienta guía paso a paso al usuario para aplicarlo:

1. Se debe seleccionar en la página el elemento a refactorizar (ver Figura 4.8). Como cada CSWR solo puede aplicarse en elementos de un cierto tipo, UX-Painter solo permite seleccionar aquellos que sean del tipo requerido por el refactoring. En este caso, los tipos de elementos permitidos son únicamente los formularios.

2. El CSWR a aplicar puede requerir del usuario el ingreso de ciertos parámetros adicionales. Para *Add Inline Form Validation*, se debe seleccionar también cada uno de los campos que son obligatorios (Figura 4.9).

3. Por último, se muestra una vista previa del CSWR aplicado en la que el usuario puede confirmar o cancelar los cambios (ver Figura 4.10). Allí se puede observar que cada vez que el usuario abandona un campo obligatorio sin completarlo, se agrega a éste un borde rojo indicando que el mismo debe completarse.

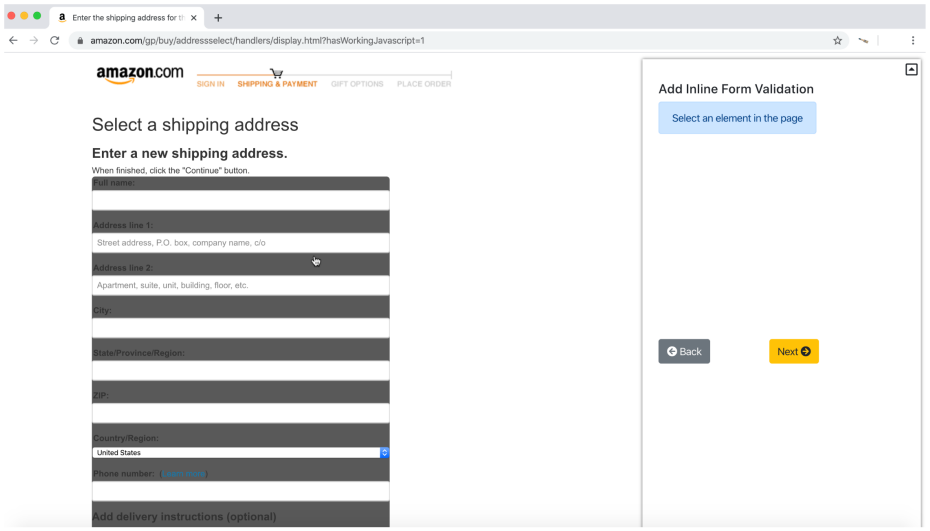

Figura 4.8: Selección del elemento a refactorizar. Solo se pueden seleccionar los elementos refactorizables con el CSWR elegido.

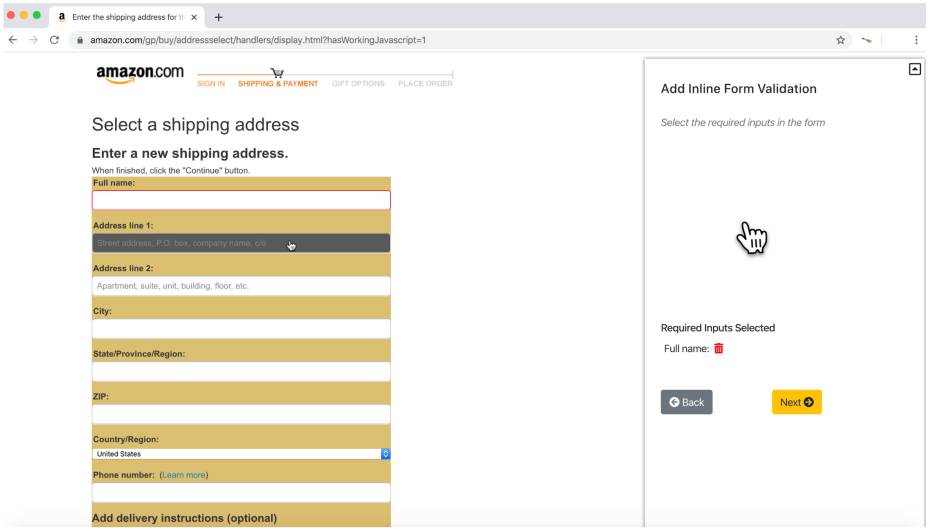

Figura 4.9: Parámetros adicionales de Add Inline Form Validation. El usuario debe seleccionar los campos requeridos.

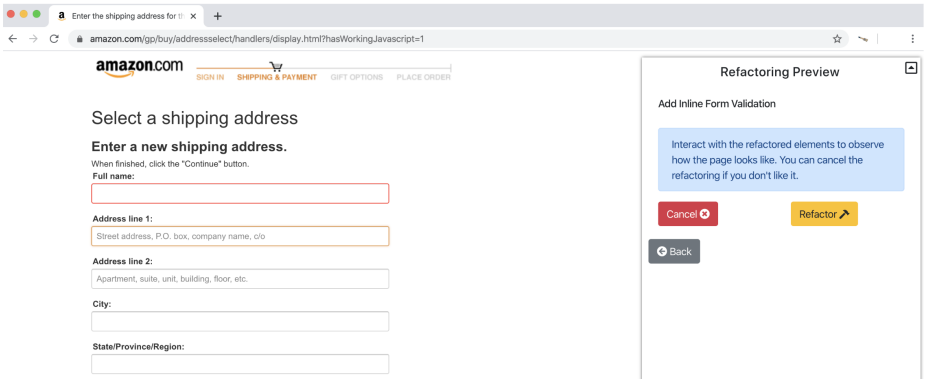

Figura 4.10: Vista previa de Add Inline Form Validation.

Después de confirmar el refactoring, se lo puede observar en la lista de refactorings aplicados en la versión actual, con la salvedad que aparece marcado con un asterisco para indicar aún no fue guardado en la versión. Para no perder este CSWR, la versión debe salvarse antes de abandonar la página actual.

Siguiendo el proceso anterior, nuevos CSWRs pueden aplicarse en la versión creada para generar cambios de diseño mas grandes. En el formulario modificado previamente, también se puede observar que el menú desplegable "Country/Region"posee una gran cantidad de opciones, lo cual puede generar que el usuario pierda mucho tiempo buscando la opción deseada. Una posible mejora para este campo es reemplazarlo por un campo de texto libre con auto-completado, mediante el CSWR *Turn Select into Autocomplete*. Este refactoring no requiere el ingreso de parámetros adicionales, por lo que luego de elegir el elemento a refactorizar, directamente se muestra la vista previa. En la vista previa, para aquellos refactorings como éste que agregan o reemplazan elementos en la UI, UX-Painter permite elegir diferentes estilos para los nuevos elementos. La herramienta ejecuta el algoritmo de adaptación de estilos detallado en la sección 4.2.4 para generar una lista de opciones de estilos que el usuario puede inspeccionar una por una y finalmente decidir cuál aplicar. En el caso de *Turn Select into Autocomplete*, tal como se puede apreciar en la Figura 4.11, las diferentes opciones de estilo permiten cambiar el color de la lista de los valores sugeridos en el campo de texto. Por defecto se muestran cinco opciones de estilo, pero este parámetro puede modificarse.

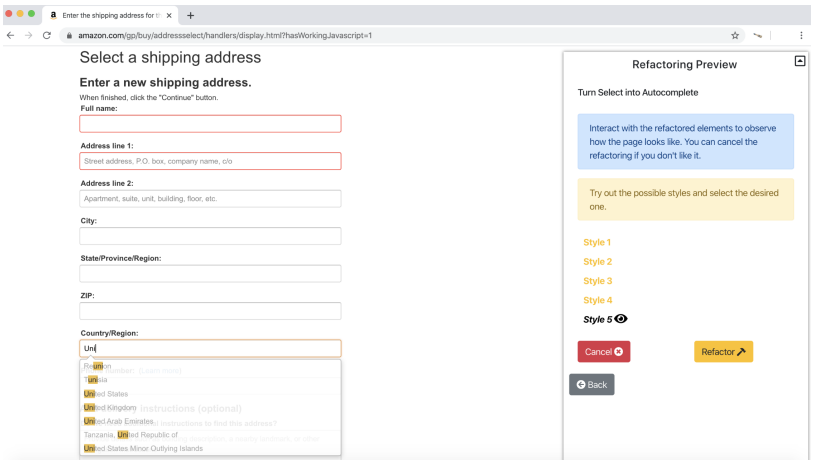

Figura 4.11: Vista previa de Turn Select into Autocomplete. Las opciones de estilo permiten cambiar el color de los valores sugeridos utilizando los colores que más se repiten en la página.

Otros refactorings que podrían mejorar el diseño de la página son *Turn Attribute into Link* para crear un link en el logo que permita navegar hacia la página principal de la aplicación, *Resize Input* en el campo "Zip"para acortar su longitud, y *Rename Element* para cambiar la etiqueta "Zip.ª "Postal Code". La Figura 4.12 muestra la versión creada que contiene todos los refactorings mencionados. En cualquier momento el usuario puede cambiar de una versión a otra seleccionándola en la lista y así comparar los resultados. Incluso una nueva versión podría ser creada para evaluar CSWRs alternativos, por ejemplo, se podría evaluar cómo funciona el formulario aplicando *Add Late Form Validation*.

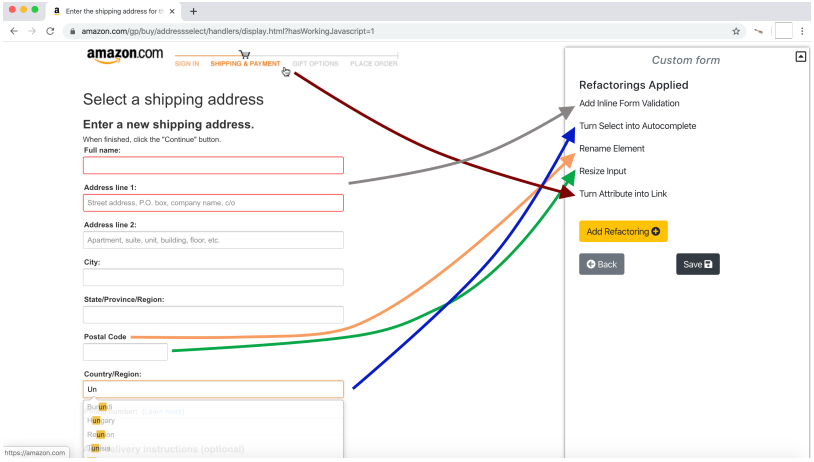

Figura 4.12: Versión alternativa con los todos los refactorings aplicados.

Otra funcionalidad que posee UX-Painter es la posibilidad de clonar versiones para facilitar la generación de alternativas de diseño con ciertos CSWRs en común, y también de exportar las versiones para poder recrearlas en otro navegador y así poder realizar por ejemplo tests con usuarios.

4.3. Generación de código de refactorings

En la sección anterior se presentó UX-Painter como una herramienta que les permite a los diseñadores de UX generar rápidamente alternativas de diseño para poder evaluarlas. La idea es que el equipo de UX utilice UX-Painter para explorar soluciones a los problemas de UX que hayan surgido, mientras que los desarrolladores implementan la funcionalidad del siguiente incremento de producto. Una vez que el equipo de UX evalúa una versión generada y comprueba que efectivamente produce una mejora en la experiencia del usuario, los cambios impuestos por los CSWRs tienen que implementarse en la aplicación destino. Esto se debe a que los CSWRs aplicados en UX-Painter son cambios volátiles que se almacenan localmente en el navegador. Además, en algunos casos los parámetros requeridos por los CSWRs son datos de prueba que luego tienen que ser reemplazados por valores reales que incluso pueden ser dinámicos. Un claro ejemplo de esto es el refactoring *Add Autocomplete*, en el que el usuario de UX-Painter ingresa solo algunos valores a sugerir necesarios para ver el resultado del refactoring, mientras que en su implementación real probablemente los valores tengan que ser recuperados de un servicio externo.

Teniendo en cuenta lo anterior, los desarrollares deben analizar cada CSWR aplicado en una versión, comprender qué modificaciones implica cada uno, y finalmente implementarlos en base a los requerimientos de la aplicación destino. Con la finalidad de facilitarle este proceso a los desarrolladores, se propone generar para una versión creada, una implementación preliminar de sus CSWRs considerando los diferentes frameworks y librerías usados hoy en día para desarrollar interfaces de usuario web (ReactJS, AngularJS, VueJS). De esta manera, la idea es que un desarrollador al momento de tener codificar un CSWR, pueda inspeccionar una implementación del mismo en el framework/librería que se esté usando para desarrollar la aplicación, y utilizarla como base para su desarrollo y así ahorrar tiempo y esfuerzo.

La posibilidad de generar una implementación para un caso particular de un CSWR se debe a que, al ser los refactorings soluciones predefinidas, lo único que cambia entre una y otra instancia de un mismo CSWR son los parámetros definidos por el usuario, por lo cual se puede crear una implementación básica para CSWR que luego se termina de completar con

los parámetros específicos de cada instancia del mismo. Más allá de esto, la implementación final de un refactoring dependerá fuertemente del código fuente de la aplicación destino, por lo cual es muy probable que el desarrollador tenga que adaptar/modificar la implementación generada a la estructura de la aplicación, o incluso completar ciertas partes que no se pueden inferir automáticamente. En ese sentido, el objetivo de generar el código para los CSWRs no es proveer una solución completa que se pueda "pegar"directamente en el código de la aplicación, sino que la idea es darle al desarrollador un punto de partida que le permita ver en detalle cómo funcionan los CSWRs y al menos obtener el código HTML y CSS asociado a estos.

Como prueba de concepto, se extendió UX-Painter con la funcionalidad de visualizar una posible implementación de las versiones creadas en ReactJS, la librería de JavaScript más usada[2] [Gardey et al., 2023]. La Figura 4.13 muestra como se ve la herramienta con esta modificación luego de crear una versión con cuatro CSWRs, en la que se puede observar que se agregó una opción en cada versión creada que permite visualizar su implementación. Al seleccionar esta opción, automáticamente se listan los componentes de ReactJS generados (ver Figura 4.14). Un componente en ReactJS no es mas que una función de JavaScript que define una pieza de la UI. La idea de los componentes es poder crear la UI a través piezas independientes y aisladas que puedan ser reutilizadas. Si bien en este caso se muestran ejemplos de ReactJS, el concepto de componente es muy similar en otros frameworks como AngularJS y VueJS.

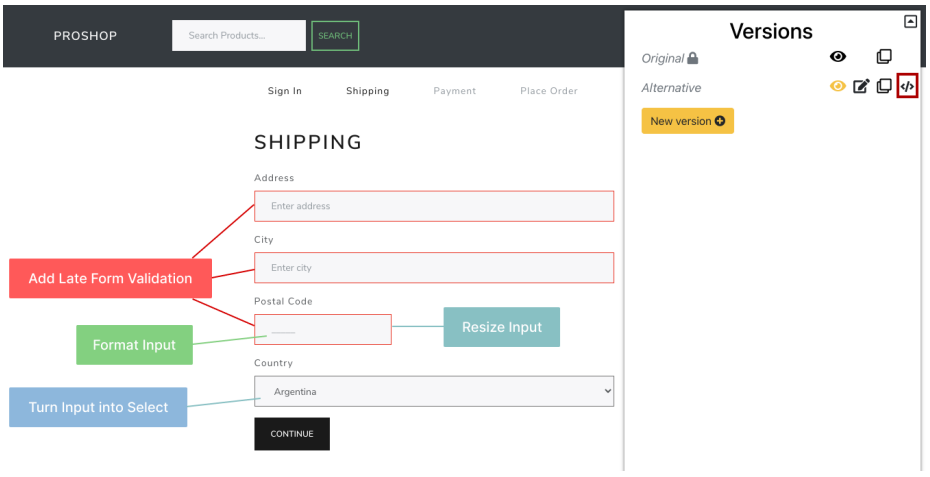

Figura 4.13: En cada versión se agrega una opción para ver una implementación de la misma.

[2]https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/

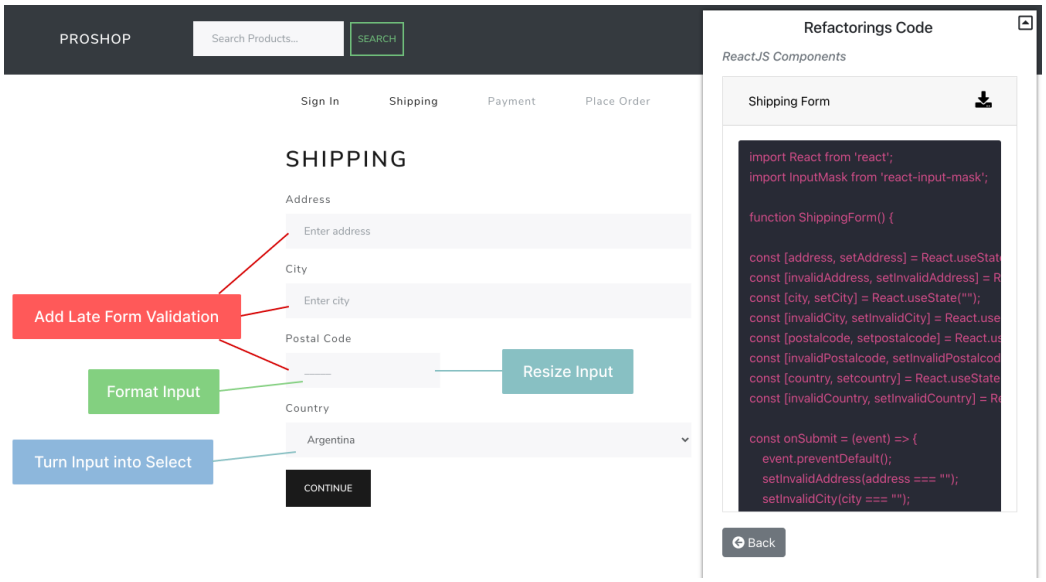

Figura 4.14: Vista del código generado para un versión en la que se listan los componentes de ReactJS.

En el ejemplo de la Figura 4.14, se puede ver que hay un solo componente generado que contiene el formulario completo. Dado que muchos de los refactorings producen cambios muy específicos, no sería útil generar un componente distinto para cada CSWR. Es por eso que la herramienta busca elementos de orden superior en el árbol del DOM (en el ejemplo es un formulario pero también podría ser una sección de página), y genera un único componente con todos los CSWRs incluidos en ese elemento.

El proceso de generación de código empieza por la creación de un componente muy básico cuyo HTML es el elemento del DOM refactorizado. Posteriormente, esta implementación se va refinando en función de los requerimientos de cada refactoring y de las especificaciones de ReactJS. Por ejemplo, en el caso de *Add Late Form Validation*, se agrega en el componente una variable de estado booleana por cada campo requerido para mostrar los errores en caso que la validación falle. Esta validación se realiza en el evento *onSubmit*, que se ejecuta cuando el usuario envía el formulario.

Una vez que se descarga el código generado, se debe integrar en la aplicación destino. Como se adelantó previamente, este proceso depende diferentes factores relacionados a la aplicación tales como: la organización del código en componentes, las dependencias o componentes externos utilizados, los requerimientos funcionales, entre otros. Incluso es probable que la implementación provista tenga que ser extendida, por ejemplo para recuperar la lista

de países de una API en el caso del *Turn Input into Select*, o para agregar otro tipo de validaciones más específicas en *Add Late Form Validation*.

El ejemplo mostrado anteriormente corresponde a un caso de prueba que se realizó en una aplicación sencilla desarrollada con ReactJS[3]. En ésta se aplicaron los refactorings mencionados antes, y luego se realizó la integración del código generado por UX-Painter en el de la aplicación, que resultó bastante simple porque el componente generado se asemeja al componente existente de la aplicación. Si bien la propuesta tiene resultados prometedores, queda pendiente realizar experimentos con aplicaciones más complejas y en entornos de desarrollo reales.

4.4. UX-Painter en un ciclo de desarrollo ágil

Como cierre del capítulo y a modo de resumen, en esta sección se describe cómo se puede utilizar UX-Painter en un ciclo de desarrollo ágil, teniendo en cuenta que el objetivo final de la herramienta es proveer un soporte para mejorar la comunicación entre los desarrolladores y el equipo de UX.

La Figura 4.15 muestra el rol de UX-Painter en las tres actividades principales de un sprint de desarrollo: *sprint planning* corresponde a la etapa en la que el equipo elige las funcionalidades a implementar en el sprint; *sprint execution* es el desarrollo del sprint en el que el equipo trabaja en las tareas seleccionadas; y finalmente *sprint review* es la reunión final del sprint en la que se inspecciona y evalúa las funcionalidades desarrolladas durante el mismo.

En este esquema propuesto, la idea es que el sprint review no solo se utilice para mostrarle el desarrollo al cliente y testear funcionalidad, sino que también se use para realizar pruebas de UX en las que se obtengan problemas o posibles mejoras relacionadas a la experiencia de usuario. Luego, durante el sprint planning de la siguiente iteración, estas incidencias se agregan al sprint backlog para ser tratadas por el equipo de UX en el sprint execution. Allí, el equipo de UX utiliza UX-Painter para explorar posibles soluciones a las incidencias surgidas, que luego podrán ser evaluadas a través de tests con usuarios o algún otro método de evaluación de UX. Mientras tanto, los desarrolladores agregan nueva funcionalidad del product backlog. En la Figura 4.15 se puede observar las actividades del equipo de UX en azul y las de los desarrolladores en naranja.

De esta manera, los desarrolladores participan en la mejora de la UX una vez que los cambios de diseño fueron evaluados. De acuerdo al proceso de la Figura 4.15, estos cambios

[3] https://github.com/bradtraversy/proshop_mern

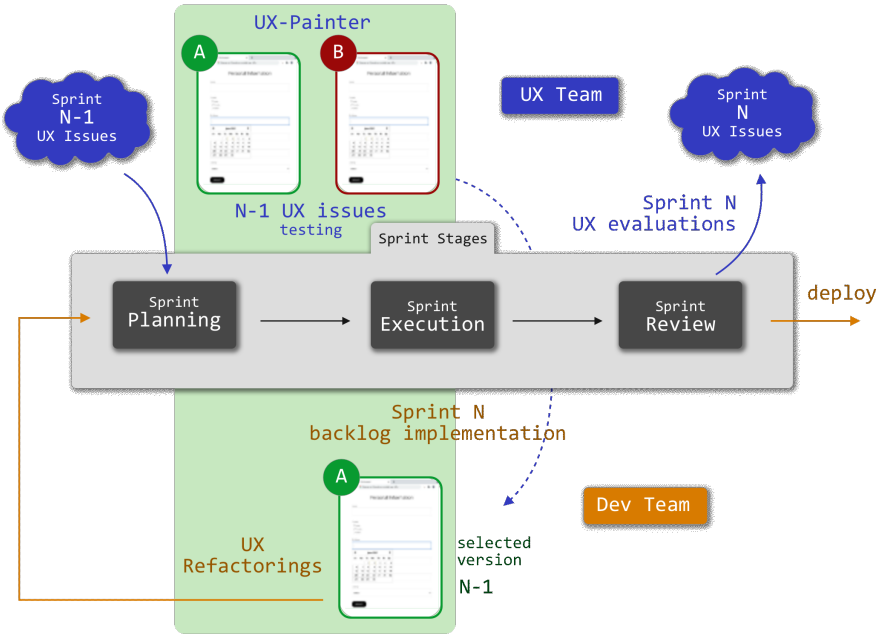

Figura 4.15: UX-Painter en un contexto de desarrollo ágil.

estarán disponibles en el siguiente sprint para implementarse, en el que durante el sprint planning, los desarrolladores pueden dividir su trabajo entre implementar los cambios o mejoras de UX evaluados y atender los ítems restantes del product backlog (que seguramente incluyen nuevas funcionalidades y corrección de errores). En cuanto a los cambios o mejoras de UX, los desarrolladores pueden usar UX-Painter para agilizar la implementación de los refactorings. Finalmente, al término de cada sprint todo el proceso comienza nuevamente con el equipo de UX realizando pruebas sobre la funcionalidad desarrollada durante el sprint execution previo.

Reducir el tiempo requerido para evaluar e implementar alternativas de diseño resulta fundamental para no posponer la mejora de la UX. Pero al mismo tiempo, también es muy importante no perder de vista el objetivo principal de los métodos ágiles que es realizar entregas de funcionalidad en períodos de tiempo acotados.

Capítulo 5

Métrica para medir el impacto de los UX refactorings

En el capítulo anterior se presentó la herramienta UX-Painter que permite explorar diseños alternativos a través de CSWRs. Estos diseños se generan mediante la creación de versiones alternativas de la aplicación en las que se aplican diferentes refactorings con los cambios de diseño deseados. Tal como se describió previamente, las versiones de UX-Painter no son modificaciones definitivas, sino que son cambios volátiles que tienen por objetivo facilitar la visualización y evaluación de alternativas de diseño. Esto significa que en un momento posterior, alguna de las versiones tiene que implementarse en la aplicación para que sus cambios sean definitivos.

Sin embargo, antes de implementar los CSWRs es importante evaluarlos de algún modo para asegurarse que efectivamente produzcan una mejora en la experiencia del usuario. Si bien los refactorings fueron pensados para mejorar la UX, el impacto que cada uno tenga en los usuarios va a depender del contexto específico de la aplicación web destino. Además, otro aspecto importante a considerar es que en ciertos casos existen refactorings alternativos para un mismo problema de UX, por lo cual se necesita de un mecanismo que permita comparar las variantes y determinar cuál funciona mejor en cada caso particular.

En este capítulo se presenta una métrica propuesta para evaluar los cambios impuestos por los CSWRs. La misma se calcula automáticamente a partir de los eventos de interacción de los usuarios.

5.1. Evaluación de refactorings

A lo largo de este trabajo se describió a los UX refactorings como transformaciones predefinidas que tienen por objetivo mejorar la experiencia del usuario. Si bien se ha demostrado la efectividad de los refactorings en trabajos anteriores [Grigera et al., 2016], lo cierto es que al ser transformaciones genéricas aplicables en diferentes situaciones, puede haber casos en los que un refactoring en lugar de mejorar la UX, provoque efectos negativos. Una situación de este estilo podría darse por ejemplo en un campo de texto libre en el que el usuario debe ingresar su fecha de nacimiento. En este caso, se podría pensar que aplicando el refactoring *Date Input into Selects*, se mejora la interacción del usuario porque se previenen los errores de formato propios de un campo que no tiene ninguna restricción. Sin embargo, dado que la fecha de nacimiento es algo que se conoce de memoria, podría ocurrir que a los usuarios les resulte más cómodo escribirla (aún con la posibilidad de cometer algún error) porque les requiere mucho menos tiempo que buscar y encontrar día, mes y año en los menús desplegables.

Por otro lado, más allá de que un refactoring no produzca los efectos deseados, también hay que tener en cuenta que existen refactorings alternativos que pretenden solucionar un mismo problema de UX. Siguiendo con el ejemplo de las fechas, para facilitar el ingreso de una fecha hay tres refactorings: *Add Datepicker*, *Date Input into Selects* y *Format Input*. Entonces la cuestión a resolver en este caso es cuál de las tres transformaciones mejora la interacción de los usuarios, o mejor dicho, cuál de los tres refactorings mejora en *mayor medida* la UX, dado que podría darse el caso que todos tengan un impacto positivo. Seguramente esto va a depender del tipo de fecha que se requiera del usuario: si es una fecha para cual se necesita tener en cuenta los días de la semana, probablemente sea más apropiado utilizar un Datepicker. Mientras que si se trata de una fecha que se conoce de antemano, posiblemente convenga seleccionarla usando menús desplegables (Date Input into Selects) o escribirla previniendo errores de formato (Format Input). Más allá de las conjeturas o suposiciones que se puedan realizar, la elección del refactoring a implementar no debe hacerse sin algún tipo de prueba o evaluación que la respalde, dado que un cambio de diseño que se asume positivo puede generar una regresión en la experiencia del usuario.

Por todo lo expuesto anteriormente, es fundamental evaluar cada refactoring en el contexto de la aplicación en la cual se realiza. Esto significa que se debe evaluar cada instanciación concreta de un refactoring en función de alguna métrica que permita determinar su impacto en los usuarios y compararlo con el de otros refactorings alternativos si es necesario.

Lo primero a tener en cuenta al momento de evaluar los refactorings es que éstos modi-

fican aspectos que influyen en la interacción del usuario con la UI, tales como el reemplazo de un widget por otro para ingresar información, o el agregado de ciertos elementos ante una determinada acción del usuario como es el caso de la validación de un formulario. Por lo tanto, estas cuestiones no pueden evaluarse simplemente con métricas que se calculan estáticamente como las que proponen Oulasvirta et al. [Oulasvirta et al., 2018], sino que se requiere de usuarios que utilicen la UI modificada para analizar cómo estos responden a los cambios impuestos.

La desventaja que poseen las pruebas con usuarios es que suelen ser muy costosas en tiempo y recursos porque hace falta reclutar usuarios y también es necesario contar con expertos en UX no solo para diseñar y moderar estas pruebas, sino también para analizar los resultados y obtener conclusiones a partir de éstos. Si bien se han propuesto herramientas que permiten realizar pruebas remotas para facilitar el acceso a los usuarios y la moderación [Carta et al., 2011, Burzacca and Paternò, 2013], aún sigue siendo necesaria la presencia de expertos en UX para el análisis de los resultados.

Uno de los enfoques que se han propuesto para automatizar la evaluación de la UX es el análisis de logs de interacción de los usuarios que usan la aplicación [Ivory and Hearst, 2001]. La ventaja fundamental de este tipo de evaluación es que se puede realizar de forma transparente para los usuarios, es decir, sin que éstos se den cuenta de que están siendo evaluados.

En este trabajo se propone una métrica para analizar el impacto de los refactorings denominada *esfuerzo de interacción*. Esta métrica representa el esfuerzo requerido por los usuarios para interactuar con los elementos de una UI individualmente. El esfuerzo de interacción se define en base a los eventos de interacción de los usuarios. Estos eventos permiten calcular ciertas micro-medidas de interacción que luego son ingresadas en modelos de predicción para obtener el valor la métrica.

5.2. Esfuerzo de interacción

Tal como se definió en la sección 4.2.3, los refactorings son transformaciones muy específicas que reemplazan o modifican unos pocos elementos en la UI. Si bien hay algunos refactorings que afectan a más de un elemento, la mayoría produce cambios en un único elemento o *widget*. El término widget se utiliza para referirse a los elementos interactivos de una UI, tales como: links, campos de texto, menús desplegables, etc., que son los que se ven afectados por los refactorings. Estos widgets con frecuencia aparecen en formularios, que son el principal medio por el cual los usuarios ingresan información en la aplicación.

Dado que los refactorings modifican determinados widgets, al momento de evaluar el

impacto de los refactorings es importante analizar puntualmente la interacción de los usuarios con estos widgets. Sin embargo, la mayoría de las métricas existentes para evaluar UX engloban la UI completa, o incluso sesiones de usuario que involucran más de una página de una aplicación. Este tipo de métricas por un lado tienen la ventaja de proporcionar una visión general de algún aspecto de la experiencia del usuario, pero por el otro no permiten determinar en qué parte de la UI los usuarios experimentan problemas de interacción, algo que es fundamental para evaluar la efectividad de los refactorings. La métrica del esfuerzo de interacción definida pretende solucionar esta limitación, analizando cómo los usuarios interactúan con los diferentes tipos de widgets que se ven modificados por los refactorings. El esfuerzo de interacción es un puntaje asignado por los expertos en UX a la interacción de un usuario con un widget particular. Este puntaje va de 1 (poco esfuerzo) a 4 (mucho esfuerzo) y los expertos de UX lo asignan en base a su observación y análisis subjetivo de la interacción, como indica la Figura 5.1. Los aspectos a analizar pueden incluir: descripción del propósito del widget, restricciones de formato para el ingreso de datos, cantidad de interacciones que realiza el usuario, entre otros. Más allá de las cuestiones pragmáticas como la eficiencia y efectividad que están relacionadas con el concepto de usabilidad, en el análisis también se pueden incluir aspectos hedónicos como la estética y el confort del usuario. Tanto los aspectos pragmáticos como los hedónicos, influyen en la UX en general [ISO, 2019].

Figura 5.1: Esfuerzo de interacción. El experto en UX observa la interacción del usuario con un widget y decide el puntaje de la misma.

El objetivo de contar con un mismo puntaje para los diferentes tipos de widget es comparar el rendimiento de widgets alternativos que sirven para un mismo propósito. Esta métrica es necesaria para poder establecer una comparación entre los widgets de distinto tipo porque cada uno admite interacciones diferentes por parte de los usuarios, que no son directamente comparables. Por ejemplo, un menú desplegable solamente admite interaccio-

nes con el mouse, mientras que un campo de texto libre además de recibir interacciones con el mouse, también se usa con el teclado para poder escribir. De esta manera, utilizando la métrica del esfuerzo de interacción, es posible realizar pruebas con refactorings alternativos y seleccionar para implementar aquel cuyo widget resulte con un menor puntaje.

Dado que el esfuerzo de interacción lo asignan los expertos en UX en función de lo que ven, para poder evaluar un refactoring sería necesario contar un experto en UX que observe todas las interacciones de los usuarios y le otorgue un puntaje a cada una, pero hacer esto sería muy costoso con pocos usuarios e impracticable con muchos. Por lo tanto, con el objetivo de automatizar el cálculo del esfuerzo de interacción, se desarrollaron modelos de aprendizaje automático para predecir el puntaje de una interacción de usuario con un widget, a partir de micro-medidas obtenidas automáticamente de la misma interacción. La Figura 5.2 ilustra como es el cálculo automático del esfuerzo de interacción para dos widgets alternativos: campo de texto y menús desplegables.

Figura 5.2: Modelos de predicción de los widgets campo de texto y menús desplegables.

Desarrollar modelos de predicción implicó en primer lugar seleccionar los widgets en los cuales se necesitaba calcular el esfuerzo de interacción. Luego, considerando que cada tipo de widget admite distintos tipos de interacción, fue necesario definir cuáles eran las micro-medidas a usar como entrada en los modelos de predicción. El hecho que cada tipo de widget tenga su propio conjunto de micro-medidas es lo que hizo que se tenga que desarrollar un modelo de predicción para cada uno. Por último, antes de comenzar a codificar los modelos, hubo que recolectar los datos necesarios para el entrenamiento y testing, para lo cual se desarrolló un proceso de captura de muestras de interacción en el que participaron usuarios finales y expertos en UX.

Como prueba de concepto, en un principio el procedimiento anterior se llevó a cabo para dos tipos de widgets: campos de texto libre y menús desplegables [Grigera et al., 2019]. Los resultados obtenidos allí demostraron que es posible predecir el esfuerzo a partir de las

micro-medidas de interacción, lo que llevó a realizar nuevamente el procedimiento agregando otros tipos de widgets como: links, botones de radio, datepickers y menús desplegables para fechas [Gardey et al., 2022]. No solo se incorporaron más tipos de widgets, sino que además se extendió el conjunto de micro-medidas para capturar más patrones de interacción, y se automatizó gran parte del proceso de captura de datos, lo que permitió aumentar significativamente la cantidad de interacciones recolectadas y así obtener mayor variabilidad en los datos capturados.

En la siguiente sección se describe el proceso de selección de micro-medidas para cada tipo de widget. Posteriormente se explica cómo se realizó la captura de datos necesarios para entrenar los modelos, y finalmente se proporcionan los detalles de cada modelo de predicción implementado.

5.3. Micro-medidas

5.3.1. Proceso de selección

La identificación de las micro-medidas se realizó analizando interacciones con los 6 tipos de widget en los cuales se propone predecir el esfuerzo de interacción. Como se mencionó antes, dado que cada tipo de widget admite distintos tipos de interacción, fue necesario encontrar un conjunto de micro-medidas diferente para cada uno. Más allá de esto, se identificaron ciertas micro-medidas comunes a todos los widgets. En una primera etapa se definieron las micro-medidas para los campos de texto y menús desplegables [Grigera et al., 2019], y posteriormente se agregaron aquellas necesarias para links, botones de radio, menús desplegables para fechas y datepickers.

El primer paso en el proceso de selección fue obtener de trabajos existentes una serie de medidas y de patrones de interacción de usuario que sirvieron para confeccionar un conjunto inicial de micro-medidas. Posteriormente este conjunto se fue refinado en un experimento preliminar con 3 expertos en UX. Allí, los expertos observaron una serie de interacciones de usuario grabadas y seleccionaron de la lista de micro-medidas confeccionadas previamente, solamente aquellas que capturan aspectos que ellos utilizaron para decidir cuánto esfuerzo hacían los usuarios. Entre estos aspectos se destacaron los tiempos de pausas, velocidad del mouse y tiempo que demora el usuario en el widget. Teniendo en cuenta que las micro-medidas tenían que calcularse automáticamente, aquellas que requerían una interpretación como por ejemplo frustración o intención del usuario fueron descartadas.

Investigando trabajos relacionados, se encontró que el movimiento del mouse es un

aspecto ampliamente utilizado para analizar el comportamiento de los usuarios. Por ejemplo, la acción de volver a interactuar con un widget ya visitado ha sido reportada como un indicador de la incertidumbre que pueden experimentar los usuarios mientras interactúan con una UI [Dias et al., 2019], por lo cual se creó la micro-medida *Interactions* que captura la cantidad de veces que el usuario hace foco en un widget. Los tiempos de permanencia y de pausas también han sido considerados en el análisis de comportamiento [Attig et al., 2019, Hurst et al., 2007]. Las micro-medidas *Dwell Time* y *Typing Latency* capturan las pausas realizadas por los usuarios en los widgets. Asimismo se incorporó la velocidad del mouse, que se ha demostrado que puede servir para detectar confusión en el usuario [Speicher et al., 2014]. Por último, los eventos de teclado también han sido muy usados para identificar aspectos de interacción, especialmente cuando se trata de ingresar datos en formularios [Atterer et al., 2006], y por este motivo tuvieron un lugar importante en el desarrollo de las micro-medidas.

Luego de confeccionar el listado inicial de micro-medidas, 3 expertos en UX analizaron individualmente grabaciones de pantalla con interacciones de usuario y le asignaron a cada interacción con un widget un puntaje entre 1 y 4. Después de la puntuación, se observaron las grabaciones nuevamente junto a los expertos, a los que se les pidió que reflexionen acerca de los aspectos del comportamiento de los usuarios que habían tenido en cuenta para decidir el puntaje. Este análisis se utilizó para mejorar el listado de micro-medidas, donde se agregaron algunas que no habían sido contempladas y se eliminaron aquellas que no tenían un impacto aparente en los puntajes. Por otro lado, también hubo que descartar ciertas micro-medidas que no podían calcularse por cuestiones técnicas que no permiten capturar determinados eventos de interacción. Por ejemplo, el navegador no permite detectar los eventos del mouse que ocurren dentro de la lista de opciones de un menú desplegable.

5.3.2. Widgets

A continuación se describe el conjunto final de las micro-medidas asociada a cada widget analizado. En primer lugar se detallan aquellas que se capturan en todos los widgets y luego las que son específicas de cada tipo.

Micro-medidas Generales

- **Dwell+Hover time**: tiempo que el usuario permanece en el área del widget. Considera el tiempo que el cursor del mouse está en área del widget, y también el tiempo que el widget tiene el foco, aún cuando el cursor del mouse se encuentra fuera del área

del widget. El área del widget es básicamente un rectángulo que contiene el widget más un margen que permite capturar la actividad del mouse en los elementos que están alrededor del widget, como por ejemplo la etiqueta de un campo de texto. La Figura 5.3 muestra un ejemplo del área del widget.

- **Mouse Trace Length**: calcula la cantidad total de pixels recorridos con el cursor del mouse dentro del área del widget. La longitud de cada movimiento o trazo del cursor se calcula considerando la distancia euclidiana entre los puntos (coordenadas X e Y) de inicio y de fin del movimiento.

- **Hover & Exit**: se llama así a un movimiento del cursor hacía el área del widget que inmediatamente es seguido de otro movimiento similar pero en una trayectoria opuesta. Es decir que ocurre cuando el usuario entra al área del widget y sale enseguida, lo cual puede ser un indicio de confusión. La micro-medida contabiliza la cantidad veces que ocurre esta secuencia de movimientos.

- **Exit & Back**: es similar a la anterior pero en sentido contrario, es decir, un movimiento desde el área del widget hacia afuera, seguido de otro movimiento similar pero una trayectoria opuesta. Dicho en otras palabras, el usuario sale de un widget y vuelve a ingresar. Al igual que el caso anterior, también se contabilizan la cantidad de veces que ocurre esta interacción.

- **Interactions**: cantidad de veces que el usuario hace foco en el widget. Esta micro-medida se calcula diferente en cada tipo de widget. En el caso específico del campo y texto y el datepicker, se utiliza el evento de JavaScript *focus* para contabilizar las interacciones. Mientras que para el resto de los widgets, dado que no disparan ese evento, las interacciones se calculan con los movimientos del cursor; se considera una interacción cuando el cursor ingresa al área del widget y permanece allí al menos por *2 segundos*. Se decidió utilizar un umbral de tiempo para no tener en cuenta los casos en los que el cursor pasa accidentalmente por un widget, por ejemplo cuando el usuario se mueve de una sección de página a otra. De esta manera, pasado el umbral se asume que el usuario tiene la intención de interactuar con el widget en cuestión. El valor de corte se determinó analizando las interacciones preliminares que se usaron para confeccionar la lista inicial de micro-medidas.

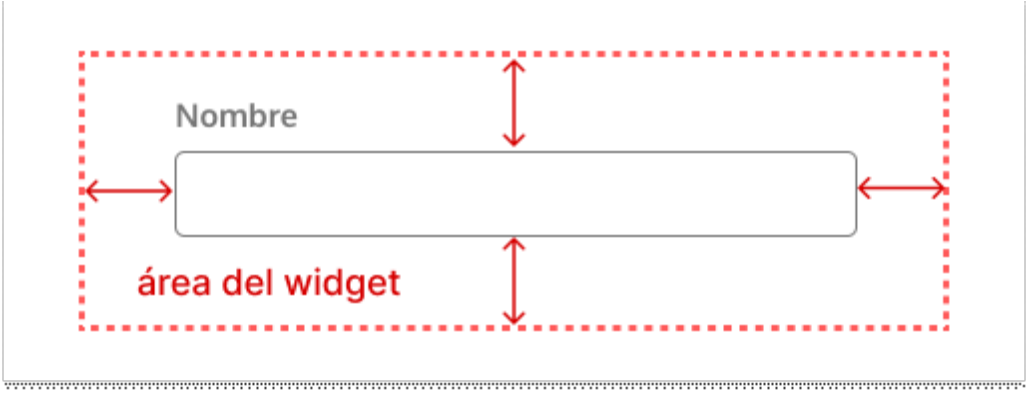

Figura 5.3: Ejemplo del área de un widget. Este área se calcula agregándole al rectángulo ocupado por el widget, un margen en cada uno de los lados (izquierda, derecha, arriba y abajo).

Micro-medidas específicas de Campo de texto

- **Focus Time**: tiempo total que el widget tiene el foco. Técnicamente es el tiempo que transcurre desde que se dispara el evento *focus* hasta el evento *blur*. Dado que esta secuencia puede repetirse, los tiempos de cada una se suman. Al igual que *Dwell+Hover Time*, esta micro-medida sirve para calcular el tiempo que el usuario permanece en el widget, con la salvedad que fue pensada para los casos en los que los usuarios utilizan el teclado en vez de el mouse para navegar entre los widgets.

- **Typing Latency**: tiempo que transcurre desde que se hace foco en el campo de texto hasta que se comienza a escribir, es decir, hasta que se pulsa la primera tecla. Si bien este tiempo esta incluído en *Focus Time*, el objetivo de esta micro-medida es estimar el nivel de duda que puede tener el usuario al interactuar con el widget.

- **Typing Speed**: tiempo de escritura en proporción a la cantidad de caracteres ingresados. Dicho de otra manera, es el tiempo que transcurre desde que se pulsa la primera tecla hasta la última dividido por la cantidad de teclas pulsadas. Esta micro-medida sirve como indicador de la eficiencia del usuario al interactuar con el widget.

- **Typing Pace SD**: es la desviación estándar de los intervalos de tiempo transcurridos entre las teclas pulsadas. Cuanto más similares sean los tiempos transcurridos entre una pulsación de tecla y otra, más baja es la desviación estándar. Por el contrario, si hay interrupciones durante la escritura, el resultado es una desviación estándar mayor.

Es por eso que esta micro-medida no tiene que ver con la velocidad, sino que apunta a cuantificar el ritmo de escritura.

- **Corrections**: es la cantidad de caracteres eliminados. Sirve como una medida de la cantidad de errores que comete el usuario cuando escribe.

- **Input Switches**: contabiliza la cantidad de cambios de teclado a mouse (o viceversa) que realiza el usuario. Esta micro-medida surgió a partir de la observación de algunos usuarios que solamente usaban el teclado para navegar por los widgets del formulario (con la tecla tab), pero cuando llegaban a un widget que no era visible se veían obligados a dejar el teclado y pasar al mouse para poder hacer scroll y así visualizar el widget. Este cambio de contexto produce una interrupción no deseada en la interacción con el widget. En este sentido, la micro-medida apunta a estimar la cantidad de movimientos, en términos de cambios en el medio de entrada, que necesita el usuario para ingresar la información requerida por el widget.

Micro-medidas específicas de Menú desplegable

- **Options Display Time**: es el tiempo total que la lista de opciones permanece desplegada. Calcula cuanto tiempo demora el usuario en encontrar y seleccionar la opción deseada. Dado que no hay manera de capturar los eventos de interacción que ocurren en la lista de opciones, el tiempo que esta lista se encuentrada desplegada se estima utilizando el tiempo transcurrido entre que se disparan los eventos *mouse down* (que ocurre cuando el usuario hacer click en el widget para ver las opciones) y *change* o *click*, ya que ambos generan que se cierre la lista de opciones: el primero ocurre cuando el usuario elige una opción y el segundo cuando el usuario hace click en cualquier parte de la pantalla fuera del widget.

- **Options Selected**: cantidad de veces que el usuario selecciona una opción en el menú desplegable. Se calcula contabilizando las veces que se dispara el evento *change* en el widget. De forma similar a *Corrections* en los campos de texto, esta micro-medida es un indicador de la cantidad de errores que cometen los usuarios (sean estos intencionales o no).

Micro-medidas específicas de Link

- **Misclicks**: es la cantidad de clicks fallidos en el área del widget. Un click fallido (missed click) ocurre cuando el usuario hace click con la intención de activar un link

específico pero la UI no genera ninguna respuesta. Básicamente son los clicks que se realizan en los alrededores del widget. Este comportamiento de hacer click donde no corresponde puede ser causado por el estilo de widget, por ejemplo cuando el área del widget no está correctamente delimitada o cuando el widget no provee un feedback adecuado acerca de cuándo puede ser clickeado.

Micro-medidas específicas de Botones de radio

Teniendo en cuenta que este widget técnicamente es una composición de elementos más pequeños, porque cada botón de radio en sí mismo es un widget, el área del conjunto de radios es la suma del área de cada elemento componente.

- **Hover to First Selection**: tiempo que transcurre desde que el cursor ingresa al área del widget hasta que se produce la primera selección. Al igual que *Typing Latency*, esta micro-medida apunta a capturar la duda o indecisión que pueden experimentar los usuarios antes de la primera interacción con el widget.
- **Selections**: similar a la micro-medida homónima del menú desplegable, es la cantidad total de selecciones realizadas. Se calcula sumando la cantidad de selecciones de cada botón de radio.
- **Misclicks**: al igual que en el caso del link, es la cantidad de clicks dentro del área del widget que no generan la selección de una opción. Esto se da por ejemplo, cuando se hace click en la etiqueta de una opción y la misma no se encuentra vinculada al botón de radio correspondiente.

Micro-medidas específicas de Datepicker

- **Selections**: es la cantidad de veces que se elige una fecha.
- **Clicks**: es la cantidad de clicks realizados en los controles del datepicker (excluyendo las selecciones de fecha). Los controles son aquellos botones que permiten navegar por el calendario cambiando el mes y el año. De esta manera, esta micro-medida sirve para estimar cuánto debe navegar el usuario para encontrar y seleccionar la fecha deseada.

Micro-medidas específicas de Menú desplegable de fecha

Este es otro caso de un widget compuesto. Dado que este widget se compone de 3 menús desplegables fijos, en este widget se utilizan las micro-medidas definidas para menús

desplegables acumulando los valores resultantes de cada componente (día, mes y año). Por ejemplo, el valor de *Selections* es 3 cuando se elige una fecha sin cometer errores.

5.4. Modelos de predicción

5.4.1. Recolección de datos

El proceso de recolección de datos consistió en 2 etapas: una primer etapa en la que se capturó una gran cantidad de interacciones de usuario, y una segunda parte en la que un grupo de expertos en UX le asignó el puntaje de esfuerzo a cada interacción. Este puntaje junto con las micro-medidas calculadas fue utilizado para entrenar y evaluar los modelos de predicción.

Para capturar las interacciones, se realizaron pruebas con usuarios en donde éstos completaron un conjunto de tareas en una serie de aplicaciones web seleccionadas. Por cada intento de un usuario de completar una tarea (también llamado sesión de usuario), se calcularon las micro-medidas correspondientes a cada interacción con un widget, y también se grabó la pantalla del usuario para que posteriormente los expertos puedan observar las interacciones al momento de asignar el puntaje.

Tanto la captura de interacciones como la asignación de puntajes se realizaron utilizando como soporte una herramienta que fue desarrollada para este fin. Contar con una herramienta de soporte y un estricto protocolo de captura no solamente permitió maximizar la cantidad de muestras, sino que además uniformizó las condiciones en las cuales se capturaron los datos, tanto para los expertos como los usuarios.

Captura de interacciones

Participaron 52 usuarios (23 mujeres y 29 hombres) en las pruebas realizadas para obtener las interacciones. Para lograr una muestra lo más representativa posible, se eligieron usuarios de diferentes edades y ocupaciones, y con distintos niveles de uso de la web. Entre los usuarios participantes había contadores, estudiantes de secundario, informáticos, médicos, maestros de jardín y jubilados.

El proceso de captura fue completamente online, todas las pruebas se realizaron en forma remota y sin ningún tipo de moderación. Cada participante recibió un enlace por e-mail en que se detallaban todos los pasos que debía seguir. Las primeras instrucciones estaban destinadas a instalar una extensión en el navegador necesaria para poder capturar las micro-medidas y la pantalla. El resto de las instrucciones correspondían a cada tarea

específica que el participante debía completar.

En cuanto a los sitios web utilizados, se eligieron 6 aplicaciones teniendo en cuenta la presencia de los tipos de widget analizados y también considerando diferentes dominios, tales como e-commerce, salud y cuidados médicos, acceso a información publica. Las aplicaciones seleccionadas tenían que cumplir 2 requerimientos: por un lado incluir la mayor cantidad de tipos de widget posible, y por el otro contener tareas que tengan sentido para todos los usuarios seleccionados.

Cada participante completó una tarea en 4 de los sitios web elegidos. Hubo 2 sitios que fueron dados a todos los participantes y los 4 restantes fueron distribuidos proporcionalmente para que cada uno tenga aproximadamente la misma cantidad de usuarios. Las tareas a completar en cada sitio fueron las siguientes:

- Buscar y agregar al carrito de compras 2 prendas específicas en un sistema de comercio electrónico de una marca de ropa[1].
- Cotizar un seguro para un modelo de automóvil determinado[2]. Para esto, los participantes completaron un formulario ingresando su información de contacto (teléfono y e-mail), y los datos del automóvil que incluían la patente, marca, modelo y versión.
- Obtener un turno médico en la aplicación web de una clínica[3]. Esta tarea también se trató de un formulario que los participantes debían completar con sus datos personales, la ciudad donde requerían el turno, la razón por la cual se solicita el turno y las fechas en la cuales el solicitante contaba con disponibilidad para asistir.
- Realizar una reserva en el estacionamiento de un aeropuerto internacional[4]. El participante debía ingresar la fecha de incio y fin de la reserva, y la patente y el tipo del vehículo (moto, auto o colectivo).
- Registrar un vehículo en el sistema electrónico de peaje[5]. Esto requería ingresar los datos personales del participante (nombre, domicilio y teléfono), el vehículo (modelo y patente), y finalmente los datos de la tarjeta de crédito para procesar los pagos a futuro.

[1] https://lacoste.com
[2] https://selfrefactoring.s3.amazonaws.com/testsites/lacaja.html
[3] https://selfrefactoring.s3.amazonaws.com/testsites/mayoclinic.html
[4] https://selfrefactoring.s3.amazonaws.com/testsites/aerolin eas.html
[5] https:// selfrefactoring.s3.amazonaws.com/testsites/telepase.html

- Obtener la constancia de CUIL (Clave Única de Identificación Laboral) en la página oficial de ANSES[6]. Allí, los participantes debían buscar la opción correspondiente y luego ingresar los datos solicitados: nombre, Documento Nacional de Identidad, género y fecha de nacimiento.

Las primeras 2 tareas fueron realizadas por todos los participantes. Esto se debió a que la búsqueda de productos en una tienda online fue la única tarea que requería navegar dentro de la aplicación web, lo cual permitió capturar muchas interacciones con links. El resto de las tareas consistían en completar un formulario en una única página. En el caso específico de la cotización de un seguro, también se decidió que lo hagan todos los participantes porque el formulario contenía 5 de los 6 tipos de widget analizados (lo único que no había en esa página eran menús desplegables para fechas).

La primera y la última tarea fueron realizadas en los aplicaciones web reales. Para el resto de la tareas, dado que son formularios que requieren información privada, se desarrolló una réplica de los formularios para no enviar datos a las aplicaciones reales. Estas réplicas fueron creadas respetando la estructura del formulario original con todos sus campos, y se incluyeron todas sus validaciones para que los usuarios interactúen con el formulario de la misma forma que lo harían en la página original. Más allá que los participantes utilizaron páginas ficticias, se les permitió realizar pequeñas alteraciones en su información personal, como por ejemplo cambiar un dígito en su DNI o teléfono, si es que no querían ingresar sus datos verdaderos.

Tal como se adelantó antes, los participantes utilizaron una extensión web para grabar sus interacciones. Esta herramienta permitió grabar cada sesión de usuario, que consistía en una colección de *logs* con las micro-medidas de cada widget que utilizó el usuario, y una grabación de la pantalla que luego sería analizada por los expertos en UX.

La extensión web agrega un botón al lado de la barra de direcciones (ver Figura 5.4) que se utiliza para comenzar y detener una captura. Cuando se inicia una captura se activan 2 componentes: un script que calcula las micro-medidas mientras el usuario interactúa con los widgets (técnicamente llamado *logger*), y otro que graba la pantalla del usuario (*screen recorder*).

El script que graba la pantalla del usuario utiliza una librería externa denominada Rrweb[7], que genera un video de una página web utilizando como entrada su HTML y CSS, y registrando todos los eventos de interacción generados por el usuario (movimientos

[6] https//anses.gob.ar
[7] https://www.rrweb.io

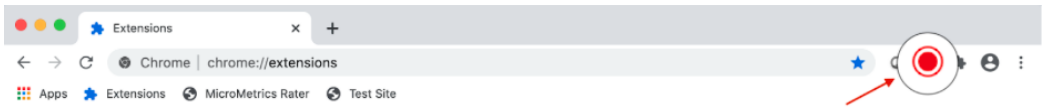

Figura 5.4: Extensión web que instalaron los participantes para capturar las interacciones.

del mouse, clicks, eventos de teclado, etc.) que posteriormente en la reproducción se re-ejecutan en forma ordenada. Se decidió utilizar este mecanismo para poder embeber tanto la grabación de pantalla como el cálculo de micro-medidas en una única herramienta, y así facilitarles la instalación del capturador a los participantes.

Asignación de esfuerzo

Para la asignación del puntaje a cada interacción recolectada, se convocó a un grupo de 4 expertos en UX (UX researchers), con más de diez años de experiencia en la industria cada uno. Al momento de realizar la recolección de datos, todos ellos habían realizado una gran cantidad de pruebas con usuarios e inspecciones de heurísticas.

Una vez que finalizó la recolección de interacciones, las sesiones de usuario generadas se dividieron en 4 subconjuntos, y para cada subconjunto se designó una pareja de expertos. La idea detrás de esta organización era que cada muestra sea puntuada por 2 expertos para minimizar posibles sesgos en el puntaje. Además, considerando que había 4 subconjuntos de muestras, para conformar 4 parejas de expertos cada uno formó pareja con otros 2, lo cual hizo que las parejas se solapen. Esto también contribuyó a que los puntajes asignados sean más uniformes.

Cada experto observó las grabaciones de pantalla de las sesiones a las que fue designado, y estableció un puntaje para cada interacción del usuario con un widget usando una herramienta desarrollada para este fin. No hubo criterios para para asignar los puntajes: cada experto solo sabía que tenía que elegir entre 4 posibles valores. Por lo tanto, los expertos asignaron el puntaje en base a su propia percepción de la interacción, sin tener conocimiento de las micro-medidas que se calcularon automáticamente. También es importante mencionar que los expertos solo veían la pantalla del usuario, no contaban con otro tipo de información como la voz o gestos del usuario. Estos aspectos no fueron tenidos en cuenta porque el objetivo era capturar las interacciones en un entorno lo más parecido posible a un contexto de uso real, en el que los usuarios no son conscientes que están siendo evaluados.

Luego de asignar los puntajes a todas las interacciones de una sesión, en aquellos casos en los que los 2 expertos asignaron diferentes puntajes, se les pidió que acordaran un valor,

para lo cual tuvieron que analizar nuevamente las interacciones y reflexionar en conjunto acerca de cuál era el puntaje más adecuado.

Los expertos utilizaron una aplicación web para establecer el puntaje de cada interacción. Esta aplicación permitía visualizar la grabación de pantalla de cada sesión, junto con las interacciones con widgets capturadas. La Figura 5.5 ilustra lo que veía el experto al momento de ingresar a una sesión de usuario: a la izquierda la grabación de pantalla y la derecha el listado de widgets con los que interactuó el usuario. Por cada widget, se puede apreciar que hay 4 botones para elegir el puntaje. Además, de cada widget se muestra su etiqueta, una letra que indica su tipo (A=link, R=radio, S=select, etc.), y cuando se ubica el cursor sobre un widget en listado se resalta en la grabación de pantalla el elemento asociado. Esto le permitía al experto encontrar fácilmente la interacción en el video. Los puntajes de cada experto se identificaban utilizando el valor ingresado en el campo "Rater ID", que generalmente era el nombre del experto.

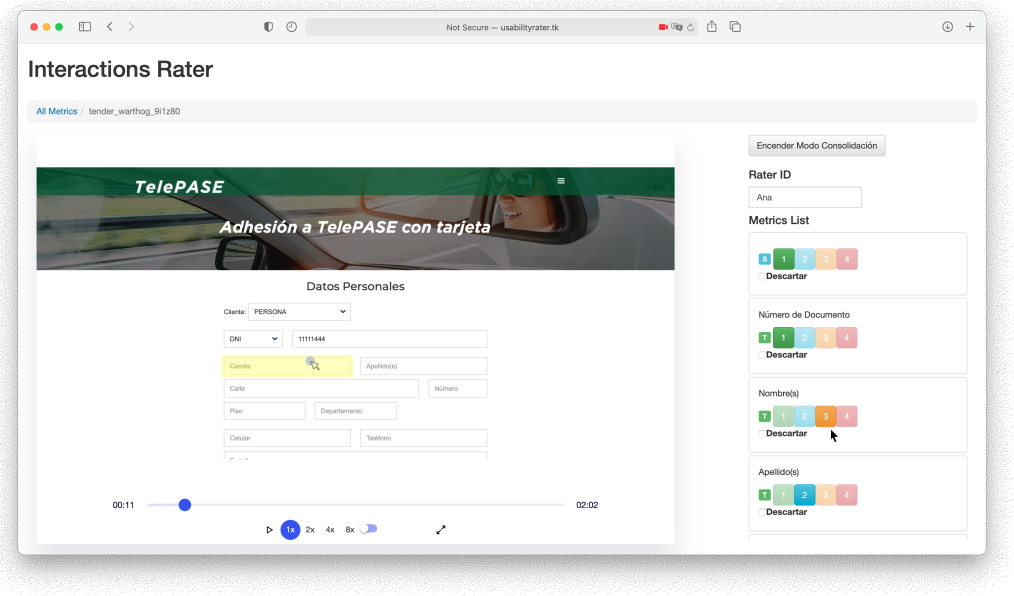

Figura 5.5: Aplicación web utilizada por los expertos para la asignación de puntajes. A la izquierda se muestra la grabación de pantalla y a la derecha el listado de widgets cuya interacción debe ser puntuada.

Si bien cada experto no tenía permitido visualizar los puntajes asignados por su par en cada sesión, la aplicación web contaba con una funcionalidad denominada "modo de consolidación"que al activarla señalaba aquellas interacciones que habían sido puntuadas

con valores diferentes, junto con el puntaje de cada experto, y permitía seleccionar el valor final utilizando la etiqueta "Consolidado". Esta funcionalidad fue utilizada por las parejas de expertos luego de establecer los puntajes en cada sesión de usuario, para acordar un valor en las interacciones que presentaban diferencias.

5.4.2. Implementación

Una vez recolectado un conjunto de interacciones (en términos de micro-medidas) con sus respectivos puntajes de esfuerzo de interacción asignado por los expertos, el siguiente paso fue desarrollar modelos de predicción para determinar la posibilidad obtener el esfuerzo de interacción automáticamente, es decir, sin que tenga que intervenir un experto en UX. En particular, teniendo en cuenta que el esfuerzo de interacción es un valor subjetivo que puede variar según el criterio de cada experto, el objetivo de este trabajo era analizar si, a partir de las micro-medidas desarrolladas, era posible predecir el esfuerzo de interacción utilizando técnicas de aprendizaje automático. Con este propósito, se desarrolló un modelo de predicción para cada tipo de widget que toma como entrada las micro-medidas de una interacción de usuario con un widget, y genera como resultado un valor que va desde 1 (poco esfuerzo) a 4 (mucho esfuerzo).

Preparación de datos

El desarrollo de modelos de aprendizaje automático comienza por la preparación de los datos que se van a utilizar para entrenar y evaluar estos modelos. Es por eso que una vez que se recolectaron las interacciones y las parejas de expertos acordaron un puntaje de esfuerzo para cada una, se llevó a cabo un proceso de limpieza de datos en el que se eliminaron aquellas interacciones cuyas micro-medidas contenían valores nulos o atípicos (conocidos como "outliers"). La Tabla 5.1 muestra la cantidad total de interacciones recolectadas de cada tipo de widget, junto con la cantidad de muestras descartadas.

Tabla 5.1: Cantidad de interacciones recolectadas para cada tipo de widget.

Widget	**Total**	**Descartadas**	**Usadas**
campo de texto	755	60	695
menú desplegable	394	27	367
Anchor	474	74	400
botones de radio	582	27	555
datepicker	171	2	169
menú desplegables de fecha	101	9	92

Los valores atípicos y erróneos se detectaron analizando las grabaciones de pantalla capturadas. Observando las interacciones, se pudieron identificar errores y discrepancias en las micro-medidas capturadas. Por ejemplo, hubo interacciones cuyo valor de la micro-medida *Dwell+Hover time* no reflejaba el tiempo observado en la grabación de pantalla, siendo este último mucho menor al valor calculado. Este tipo de diferencias probablemente se generaron por errores en el logger, o comportamientos no esperados del usuario que no eran visibles en las grabaciones. Considerando que no se pudo identificar la causa de estos errores, estas interacciones fueron descartadas para no generar sesgos en los modelos de predicción.

En cuanto a las micro-medidas calculadas, se descartaron las llamadas *Exit & Back* y *Hover & Exit* porque en la gran mayoría de los casos sus valores fueron 0, lo cual puso en evidencia que los patrones de interacción que pretendían capturar eran muy poco comunes.

Selección de modelos

Se decidió abordar la predicción del esfuerzo de interacción como un problema de regresión en vez de un problema de clasificación. En un modelo de clasificación, los cuatro valores del esfuerzo serían diferentes objetivos que no guardan ningún tipo de relación, lo que significa que por ejemplo si el valor de verdad es 1, el error del modelo es el mismo si predice 2 o 4, simplemente porque no acierta el objetivo. En cambio, el error de un modelo de regresión considera la diferencia entre el valor predecido y el real, lo cual en esta tarea de predicción importa porque los valores del esfuerzo de interacción siguen un orden, es decir, el valor 2 está más cerca de 1 que el valor 4.

Utilizando la librería *scikit-learn*[8] de Python para la implementación, se analizaron diferentes tipos de modelos de regresión tales como regresiones lineales, redes neuronales y árboles de decisión, con el objetivo de determinar cuál se desempeñaba mejor para cada tipo de widget. El rendimiento de los modelos se evaluó de acuerdo al *error absoluto medio* (conocido como MAE por sus siglas en inglés) y al coeficiente de determinación (R^2), que refleja cuán bien se ajusta el modelo a la variable que se pretende explicar. Además, para la evaluación se utilizó la técnica de *validación cruzada repetida*, que básicamente consiste en crear N particiones del conjunto de datos y desarrollar N modelos, utilizando en cada caso N-1 particiones para el entrenamiento y la partición restante para la evaluación, de manera que todas las particiones se usan una vez en la evaluación. Este procedimiento se repite k veces promediando los resultados de los modelos para obtener un resultado general. El uso

[8] https://scikit-learn.org/stable/

Tabla 5.2: Resultados de los modelos de cada tipo de widget.

Tipo de widget	**MAE**	R^2
campo de texto	0.22	0.72
menú desplegable	0.16	0.82
link	0.19	0.82
botones de radio	0.27	0.65
datepicker	0.26	0.57
menú desplegable de fecha	0.19	0.69

de la validación cruzada tiene por objetivo minimizar los sesgos que se pueden producir si se divide aleatoriamente el conjunto de datos para generar los subconjuntos de entrenamiento y evaluación.

Entre los diferentes tipos de modelo evaluados, los árboles de decisión fueron los que presentaron el mejor rendimiento para los seis tipos de widget. Los resultados se muestran en la Tabla 5.2. El error más grande (MAE=0.27) se consideró aceptable considerando que esto significa que en general si se redondea la salida de cada modelo (que es un valor continuo) a su entero más próximo se obtiene el valor de verdad. Esto significa que los modelos resultantes muestran que el esfuerzo de interacción puede predecirse utilizando las micro-medidas desarrolladas.

En cuanto a las configuraciones de los árboles de decisión, se utilizaron valores predefinidos para los hiperparámentros *profunidad máxima* y *cantidad mínima de muestras por hoja*, para minimizar la posibilidad de sobre-ajuste, que ocurre cuando un modelo está muy optimizado para los datos de entrenamiento pero no funciona bien con nuevos datos. En particular, se identificó el rendimiento más óptimo de los modelos utilizando una profundidad máxima de 5 y una cantidad mínima de muestras por hoja de 10. Además, para el caso específico de los conjuntos de datos de los campos de texto, links y menús desplegables en donde el puntaje de 1 era mucho más frecuente que los otros 3 valores, se utilizó la técnica de *undersampling* para descartar aleatoriamente un subconjunto de muestras con puntaje 1 en la etapa de entrenamiento, y así minimizar la tendencia de los modelos a predecir valores cercanos a este puntaje.

5.4.3. Importancia de las micro-medidas

Los árboles de decisión son considerados modelos de *caja blanca* porque no solamente proporcionan un resultado, sino que además permiten interpretar cómo se llega a éste. Analizando el árbol que se genera al entrenar los modelos, es posible reconstruir para un

valor de entrada determinado, la secuencia de decisiones que se toman para llegar al valor predicho. En forma más resumida, también se puede visualizar el *peso* que cada atributo (las micro-medidas en este caso) tuvo en el proceso de decisión. El peso de cada atributo suele ser un coeficiente que varía entre 0 y 1, siendo la suma de todos estos igual a 1.

Luego de obtener los resultados detallados en la Tabla 5.2, se analizó el peso que en promedio cada micro-medida recibió en los árboles de decisión entrenados, con el objetivo de establecer posibles relaciones entre la importancia que tuvieron las micro-medidas en el proceso de predicción y los patrones de interacción observados en las grabaciones de pantalla. Este tipo de análisis permite a futuro realizar un nuevo refinamiento de las micro-medidas utilizadas, y también identificar aquellos patrones de interacción de usuario que pueden generar experiencias deficientes.

A continuación, para cada tipo de widget se lista el peso de cada una de sus micro-medidas junto con su respectivo análisis. El peso de cada micro-medida corresponde a un coeficiente denominado *gini importance* provisto por la librería scikit-learn.

Campo de texto

La Tabla 5.3 muestra la importancia de gini de las micro-medidas utilizadas para predecir el esfuerzo de interacción de los campos de texto. En esta tabla se puede apreciar que las micro-medidas más relevantes son *Typing Pace SD*, *Focus Time* y *Dwell+Hover Time*. Las últimas 2 micro-medidas poseen una fuerte correlación (coeficiente de Pearson=0.83), por lo que se entrenaron nuevos modelos excluyendo uno de estos atributos y los resultados no se vieron modificados. La correlación existente entre *Dwell+Hover Time* y *Focus Time* es razonable porque ambas micro-medidas, al medir el tiempo que el usuario se encuentra en el widget, se suelen superponer cuando el usuario utiliza el mouse para hacer foco en el widget y deja el cursor dentro del área mientras escribe. Si bien hay casos en los que las micro-medidas no se superponen, especialmente cuando el usuario usa solo el teclado para navegar por los widgets, estos casos no fueron frecuentes en las interacciones observadas.

Tabla 5.3: Gini importance promedio de las micro-medidas de los campos de texto.

Micro-medida	Gini importance
Typing Pace SD	0.78
Focus Time	0.082
Dwell+Hover Time	0.058
Corrections	0.024
Typing Speed	0.008
Interactions	0.006
Mouse Trace Length	0.004
Typing Latency	0.002
Input Switches	0

El resto de las micro-medidas tuvo una importancia mínima en el proceso de decisión. Una posible explicación para esto es que con solo *Typing Pace SD* y *Dwell+Hover Time* o *Focus Time* el modelo logra separar las muestras con diferentes puntajes. Sin embargo, es importante mencionar que la mayoría de las micro-medidas que aparentan ser irrelevantes están contenidas en las más importantes, por lo cual los aspectos de interacción que pretendían capturar fueron indirectamente tenidos en cuenta. Por ejemplo, *Typing Latency* está incluida en *Focus Time* y en *Dwell+Hover Time* cuando el usuario deja el cursor en el área del widget; las correlaciones son 0.62 y 0.71 respectivamente. Algo similar ocurre con *Typing Speed*, que también presenta una correlación muy fuerte con *Dwell+Hover Time* (0.81). La fuerte correlación que presentan *Corrections* y *Dwell+Hover Time* también tiene sentido aún cuando son micro-medidas que captura diferentes aspectos, porque cuanto más caracteres corrija el usuario, más tiempo permanecerá en el área del widget. En cuanto a las micro-medidas *Interactions* e *Input Switches* que resultaron las menos importantes, no hay una correlación observable con las restantes.

Menú desplegable

En este tipo de widget, las micro-medidas más importantes resultaron ser aquellas que miden el tiempo que el usuario permanace en el widget: *Options Display Time* y *Dwell+Hover Time* (ver Tabla 5.4). La primera estima el tiempo total que la lista de opciones se encuentra desplegada, mientras que la segunda solo considera la actividad del cursor alrededor del menú cuando la lista de opciones está cerrada, porque el navegador no permite capturar eventos de interacción cuando la lista de opciones está desplegada. La importancia de *Options Display Time* está relacionada con que la mayoría de las veces los usuarios permanecen en el widget leyendo las opciones una por una hasta encontrar la

deseada, y por este motivo esta micro-medida es el factor principal para decidir el esfuerzo de interacción.

Tabla 5.4: Gini importance promedio de las micro-medidas de los menús desplegables.

Micro-medida	Gini importance
Options Display Time	0.801
Dwell+Hover Time	0.194
Mouse Trace Length	0.003
Interactions	0
Options Selected	0

Por otro lado, la poca importancia que recibió *Options Selected* podría atribuirse a que una vez que los usuarios seleccionaban una opción era poco frecuente que la modifiquen, por lo cual la mayoría de las interacciones recolectadas tenían una sola selección, y en los casos en los que la opción elegida se modificaba, esto generalmente ocurría una sola vez. Algo parecido sucedió con la micro-medida *Interactions*: la falta de variabilidad en sus valores hizo que no fuera considerada en el proceso de decisión.

Link

Dwell+Hover Time fue la micro-medida con mayor importancia al momento de predecir el esfuerzo de interacción de los links (Tabla 5.5). En las grabaciones de pantalla observadas, fue muy evidente que los usuarios ubican el cursor sobre un link mientras determinan si éste sirve o no para cumplir con la tarea en cuestión. Por lo tanto, el tiempo que toma esta determinación es un aspecto fundamental en la predicción del esfuerzo. El hecho que *Mouse Trace Length* esté correlacionada con *Dwell+Hover Time* (0.5) es razonable porque los usuarios tienden a analizar el contenido de los links moviendo el cursor, sobre todo cuando un link es textual. En cuanto a la micro-medida *Misclicks*, se observó que hacer click en los alrededores de un link con la intención de activarlo era muy poco común, lo cual puede explicar por qué esta micro-medida fue irrelevante en la predicción. Además de irrelevante, *Misclicks* era propensa falsos positivos cuando un grupo de links se mostraban juntos, dado que muchos de los clicks identificados como fallidos en un link en realidad estaba destinados a activar otro link muy cercano.

Tabla 5.5: Gini importance promedio de las micro-medidas de los links.

Micro-medida	Gini importance
Dwell+Hover Time	0.93
Mouse Trace Length	0.065
Misclicks	0.003
Interactions	0

Botones de radio

Al igual que el caso anterior, *Dwell+Hover Time* fue la micro-medida más relevante en el proceso de decisión de los botones de radio (ver Tabla 5.6). En general, se observó que los usuarios primero ubican el cursor en el área del widget y luego analizan las opciones disponibles para decidir cuál elegir. La correlación positiva de *Dwell+Hover Time* con *Mouse Trace Length* (0.53), puede explicarse considerando que los usuarios tienden a ubicar el cursor en cada opción que leen, y por eso el tiempo que demoran en elegir una opción también se ve reflejado en el movimiento del mouse.

Por otro lado, como ya se mencionó antes, la mínima importancia de *Hover to First Selection* tiene que ver con que al estar incluida en *Dwell+Hover Time*, los valores de ambas micro-medidas tienden a ser muy similares cuando el usuario abandona el widget luego de la primer selección, y esto es lo que sucedió en la mayoría de las interacciones observadas. En cuanto a *Misclicks*, sucedió algo similar a las interacciones con links: no era común observar clicks dentro del área del widget que no generen la selección de una opción. Por este motivo probablemente esta micro-medida no fue considerada en el árbol de decisión.

Tabla 5.6: Gini importance promedio de las micro-medidas de los botones de radio.

Micro-medida	Gini importance
Dwell+Hover Time	0.908
Mouse Trace Length	0.037
Selections	0.037
Hover to First Selection	0.027
Interactions	0.006
MisClicks	0

Datepicker

La cantidad de selecciones de fecha (*Selections*) fue la principal micro-medida que permitió la separación entre interacciones de diferentes puntajes (ver Tabla 5.7). Posiblemente

esto se debió a que los usuarios tuvieron que corregir la fecha elegida muy frecuentemente (el 30 % de las interacciones recolectadas posee al menos dos selecciones). La razón por la cual los usuarios tuvieron que corregir las fechas en varias oportunidades era que había datepickers con ciertas restricciones en las fechas permitidas que no eran oportunamente informadas al usuario, por lo que muchas veces se debía cambiar la fecha luego de un intento fallido de enviar el formulario. Un ejemplo de esta situación se daba en el sistema de reserva de estacionamiento en un aeropuerto, donde la fecha de inicio de la reserva era un datepicker que permitía seleccionar cualquier fecha, pero al momento de enviar el formulario se verificaba que la fecha elegida sea al menos dos días posterior a la fecha actual. Algo similar ocurría con la cantidad de días entre la fecha de inicio y la fecha de fin de la reserva, la cual se verificaba que no excediera la cantidad de días de la tarifa elegida.

Tabla 5.7: Gini importance promedio de las micro-medidas de los datepicker.

Micro-medida	Gini importance
Selections	0.73
Interactions	0.160
Dwell+Hover Time	0.070
Clicks	0.021
Mouse Trace Length	0.008

La importancia de *Interactions*, que en los casos anteriores fue prácticamente irrelevante, también fue consecuencia de las correcciones de fecha, porque cada selección de fecha en un datepicker generaba una nueva interacción con el widget. Esto también explica la fuerte correlación existente entre *Interactions* y *Selections* (0.9).

Con respecto a *Dwell+Hover Time* y *Mouse Trace Length*, vale aclarar que los movimientos de mouse dentro del calendario que genera el datepicker no se pudieron capturar por razones técnicas, por lo cual estas micro-medidas solo consideraban el campo de texto y sus alrededores como si no hubiera calendario. Esto probablemente explique por qué no había grandes diferencias en los valores de estas micro-medidas para los diferentes puntajes, teniendo en cuenta que los usuarios permanecían la mayoría del tiempo en el widget con el calendario abierto.

Por último, la micro-medida *Clicks* que pretendía estimar cuánto debía navegar el usuario para encontrar la fecha deseada, no resultó importante en el árbol de decisión. Esto tuvo que ver con que en general la fecha elegida por los usuarios estaba muy cerca (en términos de clicks) del punto de partida en el calendario que era el mes actual.

Menú desplegable de fecha

Este modelo es muy similar al del menú desplegable porque utiliza como entrada las mismas micro-medidas, con la diferencia que los valores de cada una se calcula sumando los valores de cada menú desplegable que compone el widget. Observando la importancia resultante de cada micro-medida (Tabla 5.8), se puede apreciar que al igual que en el modelo del menú desplegable, la que tiene mayor relevancia es *Options Display Time*. Sin embargo, *Dwell+Hover Time* en este caso tiene una mayor importancia porque los usuarios luego de seleccionar la fecha tienden a dejar el cursor en el área del widget para verificar la fecha elegida. *Mouse Trace Length* también ganó más relevancia debido a que los usuarios usaban el mouse para moverse de un menú desplegable a otro. Los movimientos del mouse se incrementaban cuando había que corregir alguno de los valores seleccionados.

Tabla 5.8: Gini importance promedio de los menús desplegables de fecha.

Micro-medida	Gini importance
Options Display Time	0.561
Dwell+Hover Time	0.333
Mouse Trace Length	0.105
Interactions	0
Options Selected	0

5.5. Discusión

En este capítulo se presentó la métrica *esfuerzo de interacción* que tiene por objetivo evaluar la interacción del usuario con diferentes widgets que son estándar en los formularios web. Los resultados de los modelos de predicción entrenados muestran que es posible determinar el esfuerzo de interacción utilizando un conjunto de micro-medidas calculadas a partir de los eventos de interacción de usuario.

La ventaja de esta métrica en comparación con otras propuestas que hacen un análisis más general considerando una página web completa o incluso una sesión de usuario con múltiples páginas, es que el esfuerzo de interacción permite detectar específicamente en cuáles elementos de la UI los usuarios tienen problemas. Esto resulta fundamental para poder evaluar el rendimiento de pequeños cambios de diseño como los que introducen los UX refactorings. Otro aspecto importante a remarcar es que el análisis de la interacción es completamente automático y no está sujeto a un conjunto de tareas predefinidas en la aplicación web destino. Por lo cual, el cálculo del esfuerzo de interacción puede realizarse

en la aplicación real sin que los usuarios se den cuenta que están siendo evaluados, lo que hace que la métrica pueda ser usada en un experimento A/B testing.

En cuanto al desarrollo de los modelos para predecir el esfuerzo de interacción, el hecho de utilizar modelos de caja blanca como lo son los árboles de decisión permitió analizar e interpretar cómo influye cada micro-medida en el proceso de predicción. A partir de este análisis, se puede concluir que las micro-medidas más relevantes fueron aquellas que miden el tiempo que el usuario permanece en el widget destino. Con respecto a las micro-medidas que contabilizan los errores cometidos por el usuario, resultaron menos importantes que las que miden tiempo, probablemente porque en cierta medida los errores cometidos determinan el tiempo requerido para interactuar con el widget. Por último, de las micro-medidas que resultaron irrelevantes en la predicción, se pudo concluir que capturaban patrones de interacción poco frecuentes; este era el caso de *Hover & Back* y *Exit & Back* en los links, y *Misclicks* en los botones de radio y links.

Una de las limitaciones para el entrenamiento de los modelos es la cantidad de interacciones recolectadas, en especial aquellas puntuadas con un alto esfuerzo (valores 3 y 4). Si bien en la medida de lo posible se automatizó el proceso de recolección de datos para capturar la mayor cantidad de muestras posibles, lo cierto es que no fue sencillo reclutar una gran cantidad de usuarios voluntarios, y aquellos que podrían haber provisto ejemplos de interacciones que demandan un alto esfuerzo, en muchos casos no contaban con el conocimiento necesario para instalar y utilizar la extensión del navegador desarrollada. Más allá de la cantidad de participantes, la recolección de interacciones también estaba limitada por la asignación del puntaje de esfuerzo por parte de los expertos en UX, la cual era una tarea costosa en términos de tiempo, teniendo en cuenta que los expertos tenían que observar atentamente cada grabación de pantalla (en algunos casos 2 veces para acordar los puntajes).

Las interacciones cuyas micro-medidas contenían valores erróneos o atípicos podrían haber confundido a los modelos de predicción. Por este motivo, se analizaron las muestras capturadas para encontrar y descartar aquellas interacciones con discrepancias entre lo que se ve en las grabaciones de pantalla y los valores de las micro-medidas.

Un aspecto que podría haber atentado contra la validez de los modelos de predicción era la subjetividad de los puntajes asignados por los expertos. El hecho de contar con unos pocos expertos podría generar que los puntajes se asignen de una forma particular que no considere el criterio de la mayoría de los expertos en UX. Para minimizar esta posibilidad es que se decidió formar diferentes parejas de expertos combinándolos entre sí, de manera que cada uno de ellos tuvo que discutir y acordar con 2 de los expertos restantes, los puntajes

para las interacciones que fueron puntuadas con valores diferentes.

Por último, es importante mencionar que el proceso de captura de interacciones fue completamente remoto y sin moderación, con el objetivo principal de maximizar la cantidad de muestras. Si las pruebas de usuario se hubieran realizado in-situ, probablemente se podría haber capturado más información de los usuarios como la voz o sus gestos, para que los expertos puedan determinar con mayor precisión el puntaje de las interacciones. Sin embargo, esto hubiera disminuido drásticamente la cantidad de muestras recolectadas. Por lo cual se priorizó trabajar con la mayor cantidad de muestras posible, aún cuando fue necesario desarrollar herramientas para poder hacer la captura en forma remota.

Capítulo 6

Visualización del esfuerzo de interacción

La métrica del esfuerzo de interacción fue desarrollada para poder evaluar la experiencia del usuario con los diferentes widgets de una UI, y así tener un criterio para poder analizar y comparar la efectividad de refactorings alternativos. La particularidad que tiene el esfuerzo de interacción tal como fue descripto en el capítulo 5, es que evalúa la interacción de un único usuario con un widget determinado. Más allá de poder analizar un refactoring específico, el objetivo final de este trabajo es evaluar con múltiples usuarios diferentes **versiones** de una aplicación que pueden contener más de un refactoring aplicado (ver Capítulo 4).

En este capítulo se presenta una herramienta denominada UX-Analyzer, que a partir de los modelos de predicción desarrollados previamente, permite calcular y visualizar el esfuerzo de interacción global de distintas versiones alternativas de una página web. De esta manera, es posible establecer una comparación entre estas versiones y determinar aquella que requiere un menor esfuerzo de interacción por parte de los usuarios.

6.1. Esfuerzo de interacción global

Como se detalló en el capítulo anterior, el esfuerzo de interacción surge a partir de la necesidad de evaluar la efectividad de los refactorings aplicados en una UI. Si bien la métrica no se limita al uso de refactorings, está centrada en el análisis de la interacción del usuario con los widgets de la UI, porque justamente son los elementos que se ven modificados por los refactorings. El hecho que la métrica se calcule sobre los elementos interactivos, también tiene la ventaja de poder determinar con precisión en qué parte de una UI (o mejor dicho en

cuáles elementos) los usuarios experimentan problemas de interacción. Además, el esfuerzo se obtiene para cada interacción de usuario, por lo cual también proporciona información sobre los usuarios de la aplicación.

Más allá de las ventajas que tiene el esfuerzo de interacción por ser una métrica que analiza aspectos muy específicos, el objetivo de este trabajo es utilizarla en experimentos online similares a A/B testing para poder decidir sobre un conjunto de diseños alternativos, cuál provee una mejor experiencia al usuario en un contexto específico. Más específicamente, lo que se pretende lograr es que muchos usuarios utilicen diferentes versiones de una aplicación, para determinar la que resulta en un menor esfuerzo de interacción. Esto implica:

- **Involucrar múltiples usuarios**. La idea es poder evaluar una UI con múltiples usuarios teniendo en cuenta que cuantos más usuarios participen de la evaluación, más preciso será el esfuerzo de interacción resultante. De hecho esta métrica fue desarrollada considerando que tenía que ser fácil de calcular para poder incluir en la evaluación tantos usuarios como se requieran.

- **Evaluar diseños completos**. Si bien el esfuerzo de interacción sirve para determinar la efectividad de un widget específico, en realidad con esta métrica se busca evaluar versiones alternativas de una aplicación como las generadas con UX-Painter, que pueden incluir varios refactorings. Como se describió en el Capítulo 4, una de las ventajas de los refactorings es la posibilidad de aplicarlos en secuencia para lograr cambios de diseño significativos. Por este motivo, resulta fundamental poder evaluar el funcionamiento de un grupo de refactorings.

Teniendo en cuenta lo anterior, para evaluar con múltiples usuarios una interfaz de usuario que puede contener varios widgets bajo análisis, se desarrolló la métrica de esfuerzo de interacción de una versión de una página web [Gardey. et al., 2022]. Esta métrica global se calcula básicamente promediando el esfuerzo resultante de cada interacción de usuario con los widgets pertenecientes a la versión en cuestión.

Para visualizar el esfuerzo de interacción, se desarrolló una herramienta denominada UX-Analyzer. Esta herramienta permite registrar diferentes versiones de una aplicación, y recibir de cada una las micro-medidas correspondientes a cada interacción de usuario con un widget, que posteriormente se usan como entrada en los modelos de predicción para obtener los valores de esfuerzo. Finalmente, estos valores resultantes se promedian para obtener un valor de esfuerzo global de cada versión que sirve para compararlas.

En la siguiente sección, se describe el funcionamiento general de UX-Analyzer junto con todas las funcionalidades que ofrece.

6.2. UX-Analyzer

UX-Analyzer es una herramienta que sirve para calcular y monitorear el esfuerzo de interacción de una aplicación web. Si bien esta herramienta fue diseñada para ser utilizada por miembros del equipo de UX de una aplicación, que son quienes tienen el conocimiento para interpretar lo que ven y tomar decisiones al respecto, lo cierto es que la herramienta le puede ser útil a cualquier persona dentro de un equipo de desarrollo que tenga interés en observar y controlar aspectos de la experiencia de usuario, tales como product owners, product managers, entre otros.

Se trata de una aplicación web en la que los usuarios pueden registrarse y crear **evaluaciones** para observar diferentes UIs de una aplicación. A su vez, dentro de una evaluación se pueden generar múltiples **versiones** de la aplicación, para las cuales se calcula el esfuerzo de interacción que facilita la comparación de diseños alternativos. Al registrar una versión, automáticamente se genera una porción de código de JavaScript que el usuario debe pegar en el frontend de la versión en cuestión, para capturar y enviar al servidor las **sesiones de usuario**. Cada sesión contiene la colección de micro-medidas resultantes de cada interacción con los widgets que existen dentro de la versión. Cuando UX-Analyzer recibe una sesión de usuario, utiliza los modelos de predicción para obtener el esfuerzo de interacción de cada widget, y luego actualiza el esfuerzo global de la versión.

La herramienta no solamente proporciona una visión general del esfuerzo de interacción de cada versión registrada, sino que además es posible inspeccionar los detalles de cada versión, entre los cuales se incluyen el esfuerzo global de cada widget y el esfuerzo resultante de cada sesión de usuario.

6.2.1. Evaluaciones

Una evaluación se realiza sobre un grupo de versiones de la aplicación. Teniendo en cuenta que una aplicación web puede contener un gran número de páginas que ofrecen funcionalidades muy variadas, las evaluaciones sirven para separar diferentes partes del diseño de una aplicación que se quieren analizar. El tamaño de estas partes dependerá de lo que se necesite evaluar, pudiendo incluir desde aspectos generales como ejemplo el proceso de checkout en un comercio electrónico, a cuestiones más específicas tales como la forma de ingresar una fecha de nacimiento en un formulario de registro.

La Figura 6.1 muestra la vista de las evaluaciones que es lo primero que ve el usuario después de iniciar sesión. Cada evaluación se identifica por un nombre que describe el aspecto del diseño que se está evaluando.

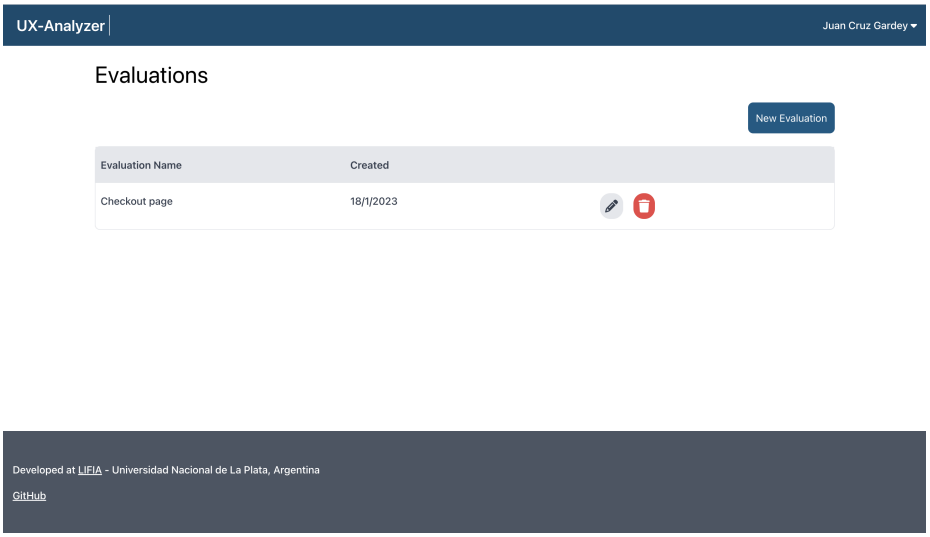

Figura 6.1: Vista principal de UX-Analyzer que muestra las evaluaciones del usuario.

6.2.2. Versiones

Las versiones de una aplicación son variantes de diseño que pertenecen a una evaluación, sobre las cuales se puede calcular el esfuerzo de interacción. Una versión también puede verse como un conjunto de sesiones de usuario capturadas en páginas de la aplicación bajo análisis. Al crear una versión, se genera el script que el usuario debe pegar en la aplicación para capturar las interacciones con los widgets. Dentro de este script, se incluye el token de la versión, el cual consiste de una cadena alfanumérica de 16 caracteres que es utilizado por la herramienta para determinar a qué versión corresponde cada sesión de usuario recibida.

La Figura 6.2 ilustra una versión recién creada. Para crear una versión, el usuario debe ingresar un nombre y una o más URLs de las páginas de la aplicación que incluye. Si bien al principio se mencionó que el esfuerzo se calcula sobre una página web, dependiendo de lo que se esté evaluando en ciertos casos puede ser necesario conocer el esfuerzo de interacción de un grupo de páginas relacionadas. Las URLs ingresadas se utilizan para filtrar las interacciones, de manera que solo se consideren aquellas que ocurren en las páginas seleccionadas.

Luego de crear una versión, en la vista de la evaluación se puede observar que se muestra una grilla con todas las versiones generadas (ver Figura 6.3). De cada versión se muestra el esfuerzo de interacción global y la cantidad de sesiones de usuario capturadas hasta el momento. Este esfuerzo global se va actualizando a medida que se reciben nuevas sesiones de usuario. La Figura 6.4 ilustra cómo se hace el cálculo del esfuerzo global en el que se promedian los valores resultantes de cada interacción de usuario con un widget. Allí se

Figura 6.2: Vista de una versión recién creada. Allí se muestra el script que el usuario debe insertar en su aplicación para capturar las interacciones.

puede apreciar que hay una interacción cuyo esfuerzo es un valor real. Esto se debe a que los modelos de predicción son regresores, por lo cual la variable predecida es siempre un valor continuo.

Figura 6.3: Lista de versiones correspondientes a una evaluación.

Al hacer click sobre una versión de la lista, se puede visualizar mejor su esfuerzo de interacción junto con otra información general de la misma (ver Figura 6.5). Además, se pueden inspeccionar los detalles de cada sesión de usuario capturada y de cada **widget** involucrado en la versión.

Figura 6.4: Esfuerzo de interacción de una versión que se obtiene promediando los valores de cada interacción de usuario con cada widget.

6.2.3. Sesiones

El script que se observa al crear una versión y que el usuario de UX-Analyzer debe pegar en su aplicación, se utiliza para calcular las micro-medidas de cada interacción de usuario con un widget. Este script es prácticamente el mismo que se usó para la captura de datos en la etapa de entrenamiento de los modelos (ver Capítulo 5), y sólo se activa en las URLs de la versión. Cuando el usuario accede a una URL de la versión, se inicia el proceso de captura que se detiene cuando se abandona la página. Por cada widget, además de las micro-medidas asociadas a su tipo, también se capturan la URL a la que pertenece y su XPath (ruta dentro del árbol DOM) que luego sirven para identificarlo unívocamente. Al finalizar la captura, se calcula la duración de la sesión de usuario y se envía toda su información al servidor de UX-Analyzer, que la procesa y la almacena en la versión de acuerdo al token recibido.

La Figura 6.6 ilustra cómo se ven las sesiones de usuario de una versión. El dato más importante de cada sesión es el esfuerzo de interacción que resulta de promediar el esfuerzo de cada widget que el usuario utilizó. Este valor se corresponde con lo que en la Figura 6.4 aparece como *Esfuerzo por usuario*. Adicional al esfuerzo, de cada sesión también se muestra su fecha y hora de inicio, duración y un identificador.

Este reporte permite tener un control de cada uno de los usuarios que interactúan con

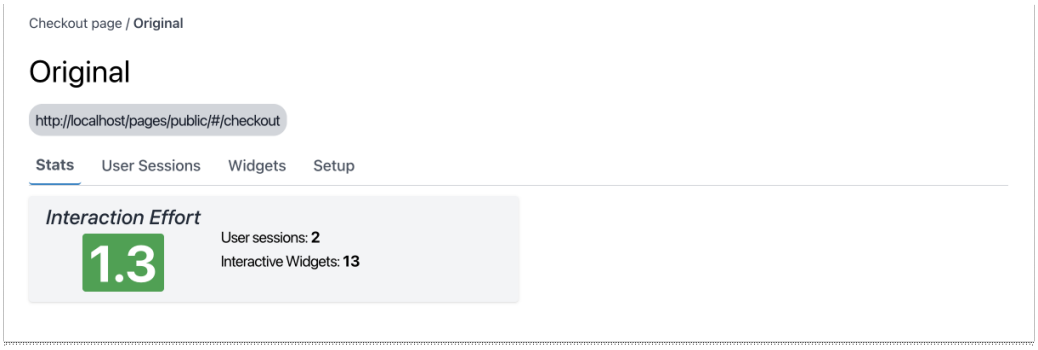

Figura 6.5: Detalles de una versión que contiene sesiones de usuario capturadas.

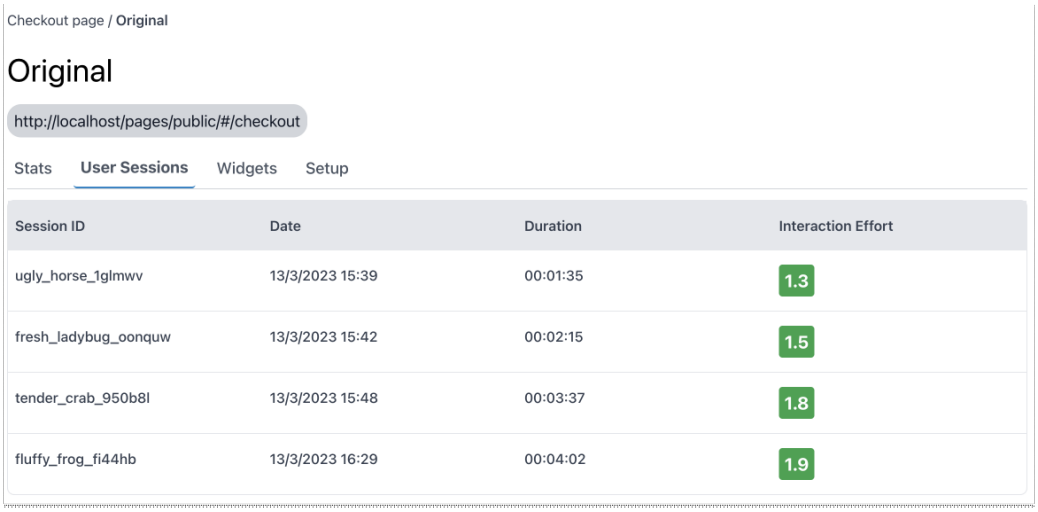

Figura 6.6: Listado de sesiones de usuario capturadas en una versión.

la versión. Analizando el esfuerzo de cada sesión y su duración, se podrían identificar por ejemplo casos particulares en los que un usuario experimenta problemas en la versión.

6.2.4. Widgets

Dentro de una versión también es posible visualizar el listado de widgets con los que interactúan los usuarios (ver Figura 6.7). Teniendo en cuenta que cada widget se identifica por su XPath y la URL en la que se encuentra, este listado se calcula agrupando todas las interacciones con widgets por estos atributos. Para cada widget se muestra el esfuerzo de interacción promedio (*Esfuerzo por widget* en la Figura 6.4), una etiqueta para que el usuario pueda identificarlo fácilmente, y su tipo (campo de texto, link, menú desplegable,

botones de radio, datepicker o menú desplegable de fecha).

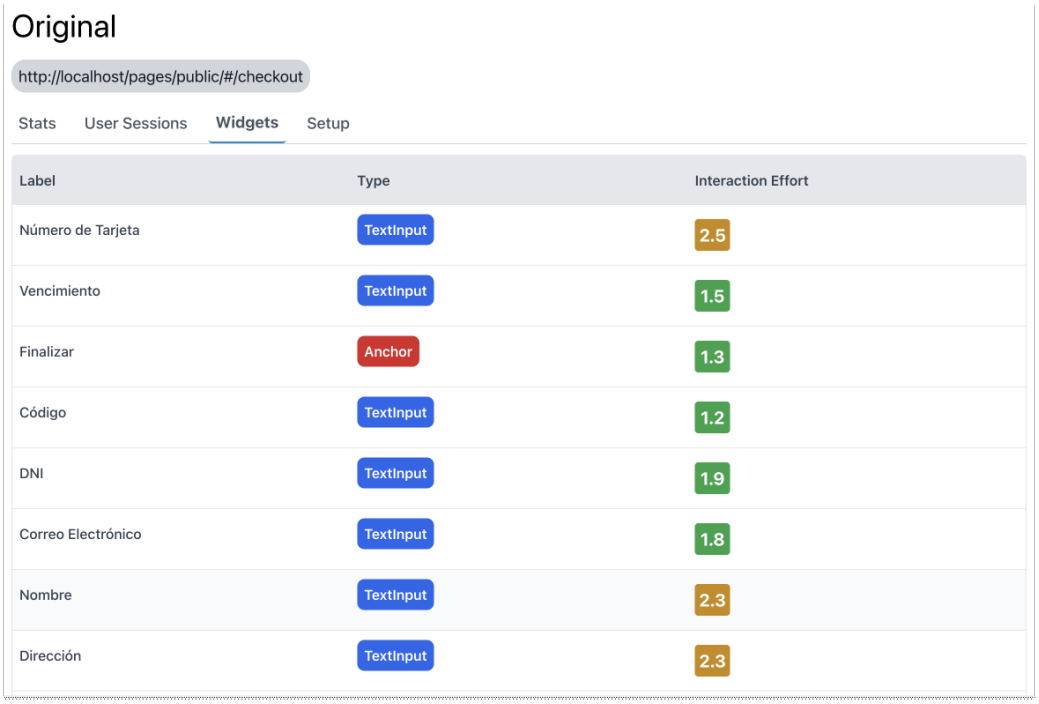

Figura 6.7: Listado de widgets incluidos en las sesiones de usuario capturadas.

Visualizar el esfuerzo de interacción de cada widget, al igual que la vista de las sesiones de usuario, también sirve para explicar cómo surge el esfuerzo global de una versión. En este caso, contar con el desempeño de cada widget permite determinar en qué parte de una UI los usuarios experimentan inconvenientes.

6.3. Discusión

En este capítulo se presentó UX-Analyzer, una herramienta para calcular y visualizar el esfuerzo de interacción de usuario de diferentes versiones de una aplicación web. Este esfuerzo de interacción se calcula automáticamente utilizando los modelos de predicción desarrollados previamente, de manera que el cálculo es completamente transparente para los usuarios de la aplicación. Esto facilita la evaluación de una UI en el contexto real de los usuarios como por ejemplo en experimentos de A/B testing, utilizando el esfuerzo de interacción para analizar únicamente aspectos de la UX sin tener en cuenta el rendimiento de un negocio. La ventaja adicional que tiene esta métrica, es que al calcularse a partir de la

interacción de los usuarios con los widgets de una UI, no solamente provee una visión global del esfuerzo de interacción, sino que además permite analizar el rendimiento cada widget por separado, lo cual sirve para determinar dónde los usuarios experimentan problemas de interacción. Este aspecto es importante destacarlo porque justamente una de las mayores limitaciones del A/B testing es que es difícil determinar el por qué de los resultados obtenidos [Nielsen, 2005].

Si bien la herramienta fue desarrollada con el objetivo de facilitar la evaluación de diseños con múltiples usuarios, su uso no está limitado al contexto de un experimento online. UX-Analyzer también podría utilizarse en pruebas con unos pocos usuarios, por ejemplo para visualizar rápidamente el impacto de un cambio de diseño o incluso para hacer una comparación de alternativas. Lógicamente, cuanto más grande sea la cantidad de usuarios involucrados, más preciso y representativo será el esfuerzo de interacción resultante. Sin embargo, es posible que si existen problemas de UX serios, los mismos se vean reflejados en el esfuerzo de unos pocos usuarios.

Con respecto a los usuarios de UX-Analyzer, la herramienta está destinada principalmente a los expertos en UX, que son quienes poseen el conocimiento para analizar e interpretar el esfuerzo de interacción resultante, y tomar decisiones de diseño al respecto. En este sentido, el objetivo de la métrica del esfuerzo no es el de reemplazar al experto en UX, sino que se trata de proporcionarle una herramienta más que pueda utilizarla oportunamente en el proceso de mejora de UX. Más allá que UX-Analyzer haya sido pensada para los expertos en UX, el hecho que sea una herramienta automática que evalúa un aspecto de la experiencia de usuario, hace que también sea apropiada para equipos de desarrollo que no cuenten con un experto en UX, o incluso para otros roles que también quieran monitorear la UX como por ejemplo líderes de proyecto, product owners, etc.

UX-Analyzer es solo una prueba de concepto para visualizar el esfuerzo de interacción de una aplicación, por lo cual a futuro se planean realizar varias mejoras. En cuanto al cálculo del esfuerzo de una versión, considerando que no todos los widgets de una UI tienen la misma importancia o reciben la misma atención por parte de los usuarios, sería bueno poder asignarle a cada widget un *peso* diferente, de manera que el esfuerzo global de una versión refleje estas diferencias. Este peso o importancia de cada widget podría ser determinado automáticamente en base a las sesiones de usuario recibidas o incluso podrían ser ajustados por el mismo usuario de UX-Analyzer. Otro aspecto a mejorar relacionado con el esfuerzo de interacción es la visualización de los distintos reportes y estadísticas que se muestran.

Más allá del esfuerzo de interacción, las mejoras más importantes de la herramienta tienen que ver con la incorporación de nuevas métricas relacionadas a la experiencia de

usuario. UX es un concepto bastante amplio, por lo cual UX-Analyzer no pretende mostrar el esfuerzo de interacción como la única métrica para evaluar UX, sino que el objetivo a largo plazo es contar con un conjunto de métricas que permitan considerar diferentes aspectos que influyen en la UX. Algunos ejemplos de otros aspectos a tener en cuenta son: las propiedades estéticas de una UI [Oulasvirta et al., 2018, Dou et al., 2019], que establecen cómo se distribuyen los elementos e incluyen cuestiones de estilo como el uso de colores, y los factores emocionales de los usuarios de la aplicación tales como satisfacción o *engagement*.

Capítulo 7

Evaluación

Este capítulo describe las evaluaciones realizadas a lo largo de este trabajo. Estas evaluaciones se dividen en 2 grandes partes: por un lado las evaluaciones correspondientes a UX-Painter que incluyen una validación del método propuesto con diseñadores y una evaluación de usabilidad de la herramienta, y por otro lado las evaluaciones relacionadas con el esfuerzo de interacción que se basan en la comparación de la métrica propuesta con otras ya existentes.

7.1. UX-Painter

7.1.1. Evaluación del método

Con el objetivo de determinar la utilidad del método desarrollado en UX-Painter para explorar y evaluar diferentes variantes de diseño, se realizaron entrevistas con 16 profesionales que trabajan en la industria: 13 de ellos eran diseñadores de UI/UX mientras que los 3 restantes eran desarrolladores de UI. Dentro de los diseñadores, los participantes ocupaban diferentes roles tales como UI/Visual designer, interaction designer y UX researcher. Las entrevistas tenían 3 partes: una primera parte con 3 preguntas cuyo fin era identificar las prácticas y herramientas utilizadas por los participantes para explorar cambios de diseño, una segunda parte en la que se le mostró a cada entrevistado una pequeña demostración de UX-Painter con un ejemplo real, y por último, por medio de 3 preguntas adicionales, los participantes reflexionaron acerca de la aplicabilidad de la técnica propuesta por UX-Painter en sus prácticas de desarrollo habituales.

A continuación se describe cada una de las preguntas realizadas junto con los comentarios más relevantes hechos por los entrevistados:

¿Cuán frecuente es que se modifique una UI que está producción para mejorar la UX?

- *Nunca*
- *Poco frecuente*
- *Frecuente*
- *Muy frecuente*

La frecuencia con la que una UI tiene que cambiarse en respuesta a un problema de UX identificado, resultó muy alta (ver Figura 7.1). Algunos de los entrevistados comentaron que si bien validan el diseño de la UI antes de su desarrollo (por ejemplo en prototipos), suelen surgir aspectos a mejorar durante y después de la implementación, ya sea porque el cliente no está convencido con el resultado o porque los usuarios experimentan problemas no identificados antes. Esto último permite resaltar la importancia de realizar evaluaciones en la aplicación real, el principal fin de UX-Painter.

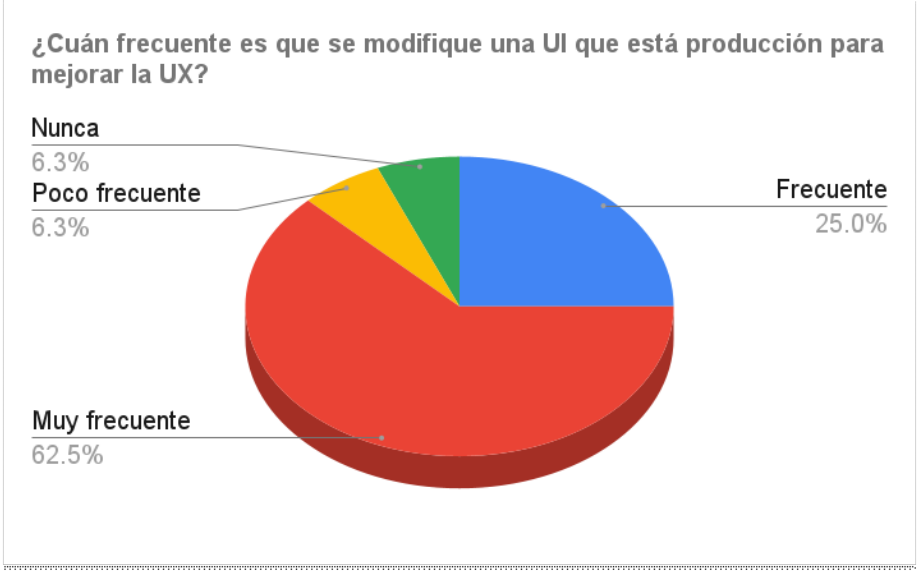

Figura 7.1: Frecuencia de modificación de una UI para mejorar la UX.

Hubo un solo participante que respondió "Nunca." esta pregunta, pero su respuesta tuvo que ver con la falta de recursos en los proyectos que ha trabajado, que hizo que sea difícil mejorar la UX luego de la etapa de implementación.

Ante un problema de UX, ¿se evalúan soluciones alternativas?

- *Nunca*
- *Poco frecuente*
- *Frecuente*
- *Muy frecuente*

En esta pregunta, 11 entrevistados respondieron que muy frecuentemente evalúan alternativas de diseño frente a un problema de UX. Sin embargo, esto no significa que siempre realizan evaluaciones formales para determinar cuál de las alternativas que consideran es mejor. A partir de las respuestas de los participantes se puede concluir que generalmente se utilizan prototipos para evaluar cambios, pero muchas veces ocurre que por cuestiones de tiempo no se llegan a evaluar todas las variantes contempladas, y por lo tanto se termina implementando aquella que se asume que funciona mejor.

¿Cómo evalúa un cambio en una UI ya desarrollada?

- *Usando un prototipo*
- *En la misma UI, implementando los cambios.*
- *Otra*

Esta pregunta apuntaba a determinar si los participantes exploraban cambios de diseño directamente en la aplicación real. Como resultado, solamente 5 de los entrevistados indicaron que realizan pruebas en la UI real usando las herramientas de inspección provistas por el navegador (ver Figura 7.2). Sin embargo, un aspecto interesante para destacar es que uno de estos diseñadores manifestó que los cambios que puede aplicar usando estas herramientas son limitados, debido a que muchas veces necesita de la asistencia de un desarrollador para inspeccionar modificaciones complejas que requieren conocimientos de programación. Además del navegador, 2 de estos 5 entrevistados también indicaron que pueden llegar a usar algún tipo de prototipo para explorar modificaciones dependiendo de su magnitud.

Más allá de lo anterior, el resultado más relevante de esta pregunta es que hay una mayoría de diseñadores (11 de 13), que únicamente usan prototipos para explorar cambios de diseño, lo cual evidencia que en general los diseñadores trabajan en artefactos separados al momento de evaluar el impacto de estas modificaciones.

Figura 7.2: Medios utilizados para evaluar los diseños.

Demostración de UX-Painter

La demostración de la herramienta consistió básicamente en mostrar un video[1] de 3 minutos de duración con un caso de uso real en el que se crean 2 versiones de una aplicación web utilizando refactorings alternativos. Luego del video, se mostraban aspectos y ejemplos puntuales de UX-Painter a pedido del entrevistado, con el objetivo de despejar todas sus dudas acerca de la herramienta.

¿Considera que el método propuesto para crear diseños alternativos puede facilitar la evaluación de la UX en el ciclo de desarrollo? ¿Por qué?

Un total de 13 entrevistados consideraron que este método puede ser muy útil para inspeccionar cambios en la UI ni bien se desarrolla, o incluso una vez que se encuentra en producción para solucionar problemas experimentados por los usuarios. Uno de estos participantes manifestó explícitamente que esta idea puede ser "más rápida y realista que un prototipo", refiriéndose a la posibilidad de evaluar los cambios directamente en la aplicación real. En cuanto a la forma de hacer las modificaciones por medio de refactorings, los participantes destacaron la velocidad con la que se aplican los refactorings y la posibilidad de contar con refactorings alternativos para un mismo problema de UX. Como punto negativo, 2 diseñadores mencionaron que los refactorings proveen cambios muy específicos para formularios.

Con respecto a los 3 participantes restantes que no consideran que este método pueda

[1] https://youtu.be/XKYrkFcNRKc

facilitar la evaluación de la UX, indicaron que ven muy difícil la posibilidad de incorporar una nueva herramienta a un flujo de trabajo que ya tienen bien definido, y en el que usan principalmente prototipos como medio para validar la UI.

¿Qué limitaciones que encuentra en la herramienta/método?

Las limitaciones mencionadas por los participantes tienen que ver principalmente con el catálogo de refactorings provisto. Los diseñadores comentaron que puede haber cambios no soportados por la herramienta que necesiten aplicar, como por ejemplo, cambios en los estilos de una página (colores, fuentes, etc.), u otras modificaciones que impliquen algún tipo de transformación en el layout de la página para mejorar la navegación, como el agregado, movimiento o eliminación de ciertos elementos. Otra limitación mencionada es que puede haber aplicaciones en donde los refactorings no se integren correctamente y produzcan efectos no deseados en la UI. Si bien esto es una cuestión importante a tener cuenta, esto tiene que ver con la implementación de los refactorings y no con la utilidad general del método propuesto que era el foco de esta pregunta.

Teniendo en cuenta que la pregunta anterior apuntaba a que los participantes reflexionen acerca de la aplicabilidad del método propuesto para evaluar la UX en general, 2 entrevistados mencionaron como limitación que UX-Painter solo considera a los usuarios interactuando con una UI, mientras que la UX es un concepto más general que va más allá de eso. Sin embargo, esto no constituye una limitación real si se tiene en cuenta que objetivo principal de este método es facilitar la evaluación de una UI.

¿Qué mejoras le haría a la herramienta?

Entre las posibles mejoras que se le podrían hacer a UX-Painter, 2 diseñadores mencionaron la de modificar el flujo de aplicación de un refactoring para que primero se elija el elemento a refactorizar y luego el refactoring. El fundamento que dieron es que el catálogo de CSWRs puede crecer, por lo cual si primero se elige el elemento a modificar, los usuarios van a tener una mejor noción acerca de los refactorings que se pueden aplicar a cada tipo de elemento. Otro aspecto a mejorar en la herramienta es la vista previa de un refactoring donde se muestran las diferentes opciones de estilo: en algunos casos estas opciones resultaron confusas porque no era fácil distinguir sus diferencias. Por último, entre las funcionalidades nuevas que se podrían incorporar, se mencionó la posibilidad de aplicar un refactoring a múltiples elementos del mismo tipo para evitar aplicaciones repetitivas, y poder visualizar el código ejecutado por cada CSWR para facilitar su posterior implementación.

7.1.2. Evaluación de usabilidad

Aparte de las entrevistas con diseñadores y desarrolladores para validar el método propuesto por UX-Painter, también se llevaron a cabo una serie de pruebas de usuario con el fin de evaluar la usabilidad general de la herramienta. La evaluación se realizó con usuarios no expertos en UX, ya que al no necesitar conocimientos de programación, un usuario no experto que recibe la instrucciones de un refactoring a aplicar, debería poder usar la herramienta sin problemas. La expertise en UX no debería influir en el uso de UX-Painter, solo le da al usuario el conocimiento necesario para decidir cuando aplicar cada refactoring. En este sentido, la evaluación de UX-Painter con usuarios no expertos permitió poner el objetivo en su usabilidad, sin considerar los cambios que UX-Painter permite realizar.

Para las pruebas se reclutaron 14 usuarios. 7 de ellos eran abogados y estudiantes de posgrado, y los 7 restantes eran informáticos investigadores. En el grupo de abogados había 4 mujeres y todos tenían entre 30 y 40 años, mientras que en el segundo grupo eran 2 las mujeres y las edades entre 25 y 60 años. Las pruebas de usuario consistieron en darle a cada participante una aplicación web y mostrarle 3 problemas de UX junto con una posible solución para cada uno. Se decidió utilizar aplicaciones conocidas para los participantes, de manera que les resulte sencillo comprender los problemas a resolver y sus posibles soluciones. En particular, los informáticos usaron un sistema de soporte para la toma de decisiones (conocido como Decision-Support System en inglés) desarrollado por su grupo de investigación, lo cual les permitió reconocer mejoras de UX aplicables en su sistema. Por otro lado, el grupo de abogados utilizó la web de Amazon[2], uno de los sitios de comercio electrónico más populares.

Luego de entender los problemas de UX que presentaba la aplicación en cuestión y de ver una demostración muy breve de la herramienta, la tarea a realizar por cada participante era seleccionar y aplicar un CSWR para cada uno de los UX smells presentados previamente. El grupo de abogados tenía que renombrar la etiqueta un campo de formulario (Rename Element), reemplazar un menú desplegable con muchas opciones por un campo de texto con auto-completado (Turn Select into Autocomplete), y convertir una imagen estática en un link (Turn Attribute into Link). Por otro lado, el grupo de informáticos tuvo que dividir una página sobrecargada de contenido en 2 secciones (Split Page), reemplazar un menú desplegable por un campo de texto (Turn Select into Autocomplete), y por último renombrar un botón (Rename Element). Luego de finalizar la tarea, los participantes completaron el cuestionario estándar SUS para tener una medida de la usabilidad de la herramienta.

[2] https://www.amazon.com

Los resultados del cuestionario revelaron un puntaje de 85/100 ($MIN = 72{,}5, MAX = 92{,}5, SD = 5{,}3$), lo que indica que los participantes encontraron UX-Painter útil y fácil de usar. Con solamente una demostración de UX-Painter en la que se mostró la aplicación de un CSWR, los participantes pudieron comprender cómo funciona la herramienta y aplicar todos los refactorings solicitados. En general los refactorings a aplicar fueron encontrados fácilmente, a excepción de Turn Attribute into Link y Turn Select into Autocomplete que llevaron más tiempo porque sus nombres no eran familiares para los participantes. En particular, los usuarios confundían Add Autocomplete con Turn Select into Autocomplete, mientras que en el caso de Turn Attribute into Link, el término "attribute"los hacía pensar que el refactoring no era aplicable a imágenes. Estas cuestiones con los nombres y descripciones de los CSWRs coinciden con algunos de los comentarios hechos por los diseñadores en las entrevistas, como por ejemplo que el término *refactoring* no es conocido dentro del mundo de la UX. En este sentido, esta evaluación de usabilidad fue fundamental para obtener feedback para mejorar el nombre de varios CSWRs, en especial de aquellos que son aplicables a diferentes tipos de elementos como Add Tooltip, Rename Element y Turn Attribute into Link.

Más allá de la selección del refactoring correcto para un problema dado, luego de aplicar todos los refactorings algunos participantes se sorprendían por los cambios de diseño que se lograban en unos pocos minutos.

7.1.3. Discusión

Las entrevistas con diseñadores de UX y desarrolladores pusieron en evidencia la falta de mecanismos para la comunicación entre los desarrolladores y el equipo de UX, que se pretende solucionar en este trabajo mediante el método desarrollado en UX-Painter. Más del 80 % de los entrevistados indicaron que la herramienta puede ser muy útil porque permite evaluar la UX directamente en la UI real. Casi el 70 % de los participantes declaró que una UI desarrollada suele modificarse muy frecuentemente en respuesta a problemas de UX detectados en producción, pero en general se utilizan prototipos para evaluar estas modificaciones porque es el recurso más barato con el que cuentan los diseñadores. Estos resultados permitieron validar el motivo principal por el cual se desarrolló UX-Painter: la falta de soporte para evaluar cambios de diseño en la aplicación real.

Con respecto a los refactorings provistos, es cierto que la mayoría de estos son transformaciones aplicables a formularios web, con lo cual pueden ser apropiados para aplicaciones que contengan muchos formularios, pero es posible que no sean aplicables en otro tipo de aplicaciones como por ejemplo aquellas que posean un flujo de navegación complejo o que

tengan problemas de UX relacionados con la disposición de los elementos. Sin embargo, el propósito de este trabajo no era mostrar una lista exhaustiva de soluciones a problemas de UX, sino que el objetivo es evaluar el uso de los refactorings como medio para explorar variantes de diseño. Dado que se pudo validar la utilidad de UX-Painter, el próximo paso será incorporar nuevos CSWRs para contar con más opciones de diseño.

Las pruebas de usuario realizadas en UX-Painter fueron importantes para determinar varios aspectos a mejorar en la herramienta. Además de las limitaciones ya mencionadas, una cuestión a tener en cuenta es que en la implementación actual de UX-Painter, la visualización de las versiones generadas puede no funcionar correctamente en las aplicaciones SPA (Single Page Application). Esto se debe a que UX-Painter ejecuta los refactorings de la versión activa cuando el navegador carga una página, mientras que en las SPA la UI se genera modificando el DOM dinámicamente sin que se produzca una recarga de página. Es por eso que para este tipo de arquitectura, es necesario desarrollar una estrategia que permita llevar un registro de todos los cambios que se van produciendo en el DOM, y así poder determinar con precisión cuándo se debe ejecutar cada refactoring.

El hecho que UX-Painter también haya sido utilizada por usuarios no-expertos evidenció otra posible utilidad de la herramienta: que los usuarios finales puedan personalizar la UI de cualquier aplicación web. Dado que no se necesitan conocimientos de programación, los usuarios podrían usar UX-Painter para solucionar las dificultades que experimenten creando sus propias vistas de un sitio web aplicando distintos CSWRs. Teniendo en cuenta que las versiones generadas se guardan en el local storage del navegador, los refactorings realizados se ejecutarán automáticamente cada vez que el usuario acceda a la aplicación. Si bien este potencial uso de UX-Painter parece prometedor, queda como trabajo futuro su evaluación.

7.2. Esfuerzo de interacción

Como evaluación del esfuerzo de interacción, se calculó la métrica en 5 páginas web seleccionadas y se compararon los resultados con los de otras métricas conocidas, con el objetivo de identificar posibles correlaciones entre éstas y el esfuerzo de interacción [Gardey. et al., 2022]. Para calcular el esfuerzo fue necesario realizar un proceso de recolección de datos en el que se llevaron a cabo una serie de pruebas con usuarios para capturar las micro-medidas asociadas a cada interacción con los widgets de las páginas analizadas, que posteriormente fueron ingresadas en los modelos de predicción para calcular el esfuerzo de cada interacción de usuario con un widget. Finalmente, para cada página web se promediaron los esfuerzos resultantes de cada uno de sus widgets de forma de obtener el esfuerzo

de interacción global.

7.2.1. Métricas utilizadas

Una de las métricas con la que se comparó el esfuerzo de interacción es la propuesta por Sauro y Kindlund llamada **SUM** (Single Usability Metric) [Sauro and Kindlund, 2005a]. Esta métrica fue desarrollada con un propósito similar al del esfuerzo de interacción, que es el de proveer un único puntaje que facilite la evaluación y comparación de la usabilidad. Lo que motivó a los autores a crear esta métrica unificada fue la gran variedad de métricas de usabilidad que existen, que con frecuencia se vuelven difíciles de utilizar para comunicar un indicador de usabilidad a los líderes de un producto u organización. De esta manera, SUM es un puntaje que combina los tres componentes de usabilidad: efectividad, eficiencia y satisfacción. Para calcular este puntaje, primero se evalúa cada componente por separado utilizando métricas de interés, y luego los valores resultantes se estandarizan mediante técnicas estadísticas para finalmente promediar los valores normalizados. Con respecto a las métricas a utilizar, los autores sugieren utilizar el tiempo requerido para completar la tarea como medida de eficiencia (time on task), la cantidad de errores cometidos (#errors) y si la tarea fue completada o no (completion) como medidas de efectividad, y por último el puntaje del cuestionario SUS (System Usability Scale) [Brooke, 1996] como medida de satisfacción [Sauro and Kindlund, 2005b]. La Figura 7.3 ilustra como se realiza el cálculo de SUM. Es importante aclarar que a diferencia del esfuerzo de interacción, un puntaje mayor de SUM implica una mejor usabilidad.

La otra métrica que se usó como referencia es el tiempo requerido para una tarea estimado por el modelo KLM-GOMS [Card et al., 1980]. Este modelo fue desarrollado hace varios años atrás con el objetivo de estimar el tiempo que tardaría un usuario experto en realizar una tarea determinada. La idea de comparar el esfuerzo de interacción con una métrica de este tipo tiene que ver con determinar si existe algún tipo de relación entre el esfuerzo y una métrica obtenida a partir de un modelo de simulación.

Para estimar el tiempo de una tarea, KLM-GOMS provee una serie de operaciones comunes que se realizan en una UI, junto con un tiempo de realización para cada una. Ejemplos de estas operaciones son apuntar un elemento con el cursor y hacer click en un botón. De esta manera, para calcular el tiempo de una tarea básicamente se deben contabilizar estas operaciones para obtener el tiempo requerido por cada una, para luego sumar estos valores y llegar al tiempo total. Para no tener que realizar este procedimiento en forma manual, se han desarrollado diferentes herramientas que permiten hacerlo automáticamente. En este trabajo en particular se utilizó KLM-Form Analyzer [Katsanos et al., 2013], una aplicación

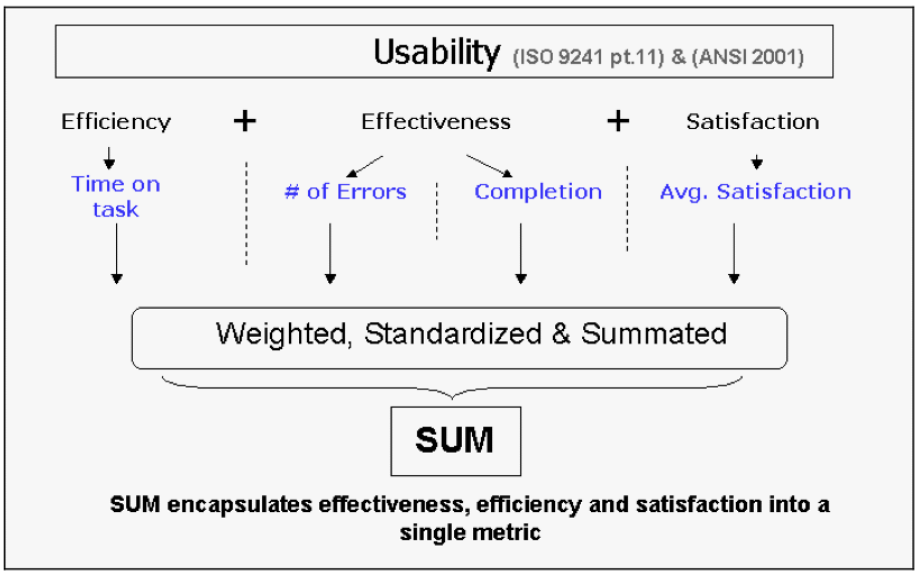

Figura 7.3: Imagen obtenida de [Sauro and Kindlund, 2005b] que resume cómo se obtiene el puntaje de SUM.

que utiliza KLM y la ley de Fitts [Fitts, 1954] para estimar el tiempo requerido para completar tareas basadas en formularios web. En esta herramienta el usuario ingresa la URL de la página a evaluar, y luego a partir del parsing automático del formulario y de ciertas configuraciones elegidas por el evaluador, se determina la lista de acciones que realizaría un usuario (operaciones) y el tiempo estimado de la tarea (ver Figura 7.4).

7.2.2. Recolección de datos

La captura de las micro-medidas necesarias para calcular el esfuerzo de interacción se llevó a cabo a través de una conjunto de pruebas con usuarios en las que éstos completaron una tarea en cada una de las páginas seleccionadas. Las páginas y las tareas realizadas en cada una fueron las siguientes:

- **Check-in de vuelo.** El usuario tenía que realizar el check-in para un vuelo ingresando sus datos personales y los datos del pasaporte provistos.
- **Turno de pasaporte.** Consistió en hacer la reserva de un turno para obtener el pasaporte argentino en alguna de las sedes de la ciudad de La Plata.
- **Checkout de compra.** Suponiendo que el usuario agregó un producto al carrito

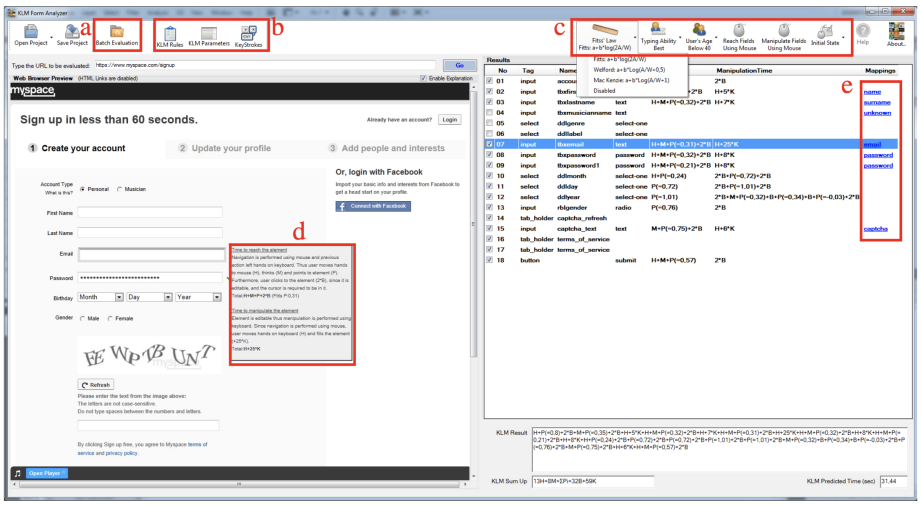

Figura 7.4: Imagen obtenida de [Katsanos et al., 2013] que muestra KLM-Form Analyzer.

de compras una tienda online, tenía que finalizar la compra ingresando sus datos personales y los datos de una tarjeta de crédito que fueron provistos.

- **Simulación de préstamo.** A partir de un tipo de cliente y una ciudad de residencia dada, el usuario debía simular un préstamo ingresando un monto y un plazo de pago.
- **Registro de usuario.** El usuario tenía que crearse una cuenta en un sistema de ventas de entradas para espectáculos ingresando su nombre, correo electrónico y una contraseña.

Estas tareas fueron realizadas por 23 participantes con edades entre 22 y 49 años y con diferentes ocupaciones. La mayoría de ellos reportó un uso diario de Internet mayor a 4 horas. Las pruebas se hicieron en forma remota y sin moderación, cada participante recibió un link con las instrucciones de cada tarea.

Las páginas web fueron replicadas para evitar el envío de datos a aplicaciones reales. Esta replicación se hizo teniendo en cuenta la estructura, los estilos y la interacción de cada página para que los usuarios puedan usarla como si fuera el sitio web real. Para capturar las micro-medidas se utilizó el mismo script que se describió en el capítulo 5, con la diferencia que en este caso el script fue embebido en cada página para no requerir la instalación de una extensión web.

Cada vez un usuario ingresaba a una página para completar la tarea, lo primero que observaba era un pequeño modal con un botón “Iniciar”que al presionarlo comenzaba con

la captura de micro-medidas y con la contabilización del tiempo de la tarea (ver Figura 7.5). La tarea finalizaba exitosamente cuando el usuario lograba enviar el formulario, o sin éxito si el usuario decidía abandonarla usando la opción "Abandonar". Luego de finalizar la tarea, sea de forma exitosa o no, el usuario tenía que responder un cuestionario para evaluar su satisfacción. Para esto se utilizó UMUX-Lite, un cuestionario de 2 preguntas que fue propuesto por Lewis et al. con la idea de eliminar la redundancia que poseen las 10 preguntas del SUS [Lewis et al., 2013]. UMUX-Lite permite evaluar la usabilidad percibida rápidamente y sin perder información, ya que un estudio posterior muestra que los resultados del UMUX-Lite están muy correlacionados con los del SUS [Lewis et al., 2015].

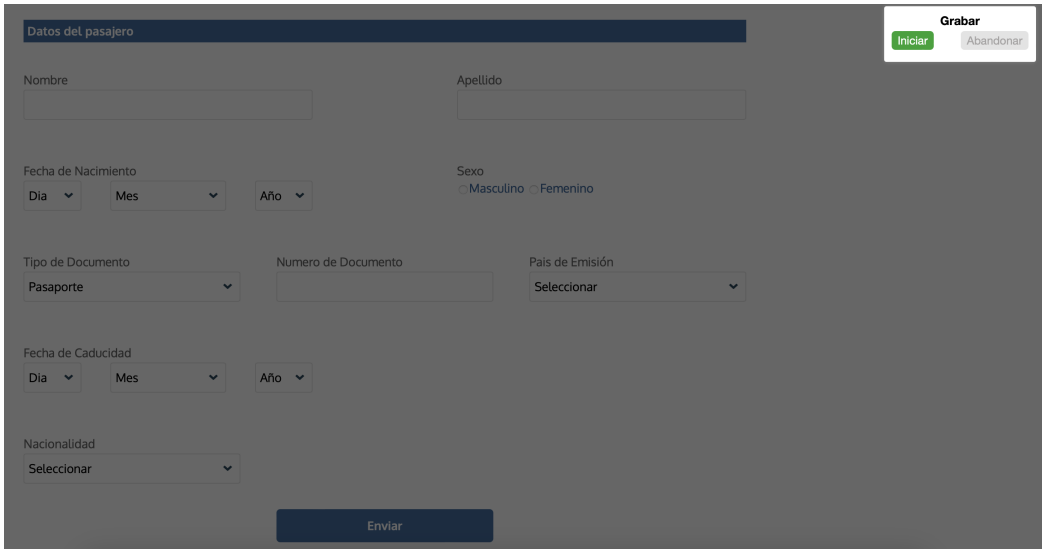

Figura 7.5: Ejemplo de una página en la que se calculó el esfuerzo de interacción, junto con las opciones de *Iniciar* y *Abandonar* la tarea.

7.2.3. Resultados

Una vez realizadas las pruebas con los usuarios, las micro-medidas obtenidas de las interacciones fueron ingresadas en los modelos para predecir el esfuerzo de cada interacción. Luego se calculó el esfuerzo global de cada página promediando los esfuerzos resultantes de los widgets que pertenecen a cada una. La Tabla 7.1 muestra los resultados obtenidos.

Para calcular el SUM de cada página, de cada tarea realizada se utilizó: su tiempo (eficiencia), si fue finalizada exitosamente o no (efectividad), la cantidad de errores cometidos en los campos del formulario (efectividad) y el puntaje del cuestionario UMUX-Lite. El valor de SUM se obtiene promediando los valores normalizados (entre 0 y 1) de las 4 medidas. El

Tabla 7.1: Esfuerzo de interacción de las 5 páginas web.

Página web	**Esfuerzo de interacción**
Check-in de vuelo	1.34
Turno de pasaporte	1.25
Checkout de compra	1.25
Simulación de préstamo	1.09
Registro de usuario	1.26

tiempo de tarea se normalizó primero calculando el valor Z restándole al tiempo promedio un tiempo estimado de la tarea y dividiéndolo por la desviación estándar, y luego ese valor Z se utilizó para determinar el porcentaje de área bajo la curva normal que corresponde al valor normalizado. Teniendo en cuenta que los autores de SUM no proveen una forma práctica de calcular el tiempo estimado de la tarea, en este caso se utilizó el tiempo arrojado por la herramienta KLM-Form Analyzer. El valor normalizado de la satisfacción se calcula de forma muy similar al anterior; obteniendo el valor Z para el puntaje promedio de las respuestas del cuestionario que se utiliza para determinar el área bajo la curva normal. La diferencia con el tiempo de tarea está en que para calcular el valor Z en la satisfacción se usa como valor de referencia el 4, que según los autores de SUM es el valor promedio que suelen tener los sistemas con una usabilidad óptima. Con respecto a la finalización de cada tarea, la normalización es muy sencilla porque solo se debe calcular la proporción de intentos finalizados exitosamente. Por último, para normalizar los errores cometidos SUM tiene en cuenta las oportunidades que ofrece cada tarea para equivocarse, dado que las tareas pueden ser muy diferentes en términos de dificultad. En este caso, los errores de cada intento de tarea se calcularon contabilizando la cantidad de campos de formulario en los que el usuario tuvo errores de validación, por lo cual las oportunidades para cometer errores corresponde a la cantidad total de campos que tiene el formulario de la tarea. De esta manera, el valor normalizado de los errores se calcula dividiendo la cantidad total de errores cometidos en todos los intentos por el total de oportunidades de error.

La Tabla 7.2 muestra el valor de SUM para cada página web, junto con los valores normalizados de las 4 medidas. Allí se puede observar que el valor de la efectividad es 1 para todas las tareas porque todos los intentos fueron finalizados exitosamente.

Para calcular el tiempo de tarea estimado por KLM, la herramienta KLM-Form Analyzer requiere la selección de algunos parámetros tales como si se aplica la ley de Fitts, y el rango etario a considerar entre ciertas opciones predefinidas. En este caso se decidió aplicar la ley de Fitts utilizando la fórmula de MacKenzie [Soukoreff and MacKenzie, 2004], y se calculó

Tabla 7.2: SUM de cada página web, que resulta de promediar los 4 valores normalizados.

Página web	**SUM**	Tiempo	Satis.	Errores	Efec.
Check-in de vuelo	0.72	0.95	0.55	0.66	1
Turno de pasaporte	0.87	0.94	0.56	1	1
Checkout de compra	0.84	0.93	0.53	0.9	1
Simulación de préstamo	0.87	0.77	0.72	0.99	1
Registro de usuario	0.72	0.95	0.42	0.34	1

el tiempo estimado para 2 grupos etarios (¡ 40 años y entre 40 y 65 años) para tomar como valor final el promedio entre ambos. La Tabla 7.3 muestra los resultados de cada página web. Teniendo en cuenta que las tareas en cada página tienen longitudes diferentes, en la tabla también se muestra la cantidad de campos de cada formulario (#widgets), y el tiempo dividido por esta cantidad para normalizar de algún modo el tiempo y así facilitar la comparación entre las páginas.

Tabla 7.3: Resultados de KLM junto con la cantidad de widgets de cada formulario y el tiempo normalizado por esta cantidad.

Página web	**tiempo KLM**	#widgets	tiempo/#widgets
Check-in de vuelo	37.6”	11	3.42”
Turno de pasaporte	9.39”	4	2.34”
Checkout de compra	51.6”	12	4.3”
Sim. de préstamo	14.06”	5	2.8”
Registro de usuario	41.3”	9	4.5”

La Figura 7.6 muestra un gráfico que compara los valores de esfuerzo con los de SUM. Para facilitar esta comparación, los valores de esfuerzo fueron normalizados entre 0 y 1 utilizando como valor mínimo el 1 y como máximo el 4. A partir de este gráfico, se puede observar que las páginas que tienen el valor de SUM más bajo (Check-in de vuelo y Registro de usuario) tienen también los valores de esfuerzo más altos. Lo contrario ocurre con la página de simulación de préstamo, que posee el valor de SUM más alto y el esfuerzo más bajo. Estos resultados sugieren que *a menor esfuerzo de interacción mayor usabilidad (y viceversa)*, lo cual en cierta medida era lo que se esperaba desde un principio. En cuanto a las páginas que no siguen esta tendencia (Turno de pasaporte y Checkout de compra), si bien tienen valores de SUM y esfuerzo altos, si se analizan los valores componentes del SUM se puede notar que los valores de satisfacción son relativamente bajos. Por lo cual, las páginas con mayor esfuerzo también pueden asociarse con menores niveles de satisfacción.

Con respecto al tiempo estimado por KLM, teniendo en cuenta que el tiempo total

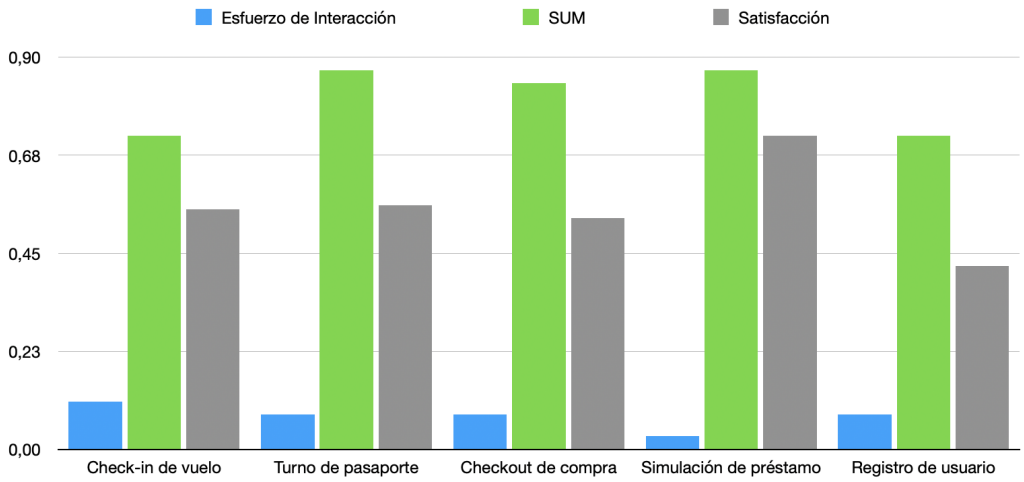

Figura 7.6: Gráfico que compara los resultados de esfuerzo de interacción (normalizados) con los de SUM. Adicionalmente, se muestran los niveles de satisfacción.

de cada página depende de la longitud del formulario, para hacer una comparación con el esfuerzo de interacción se utilizó el tiempo promedio por widget (columna tiempo/#widgets en la Tabla 7.3). Analizando los resultados, no se identificaron casos en los que se pueda establecer una relación entre los diferentes tiempos estimados y los valores de esfuerzo.

7.2.4. Discusión

Los resultados obtenidos en las páginas analizadas mostraron que un mayor esfuerzo de interacción se traduce en una peor usabilidad (en términos del valor SUM), y lo mismo ocurre en forma contraria, ya que un menor esfuerzo coincide con una mejor usabilidad. Esta aparente correlación que existe entre el esfuerzo y SUM, permite al menos utilizar el esfuerzo de interacción como métrica de usabilidad. La gran limitación de SUM es que es muy difícil obtenerla en el entorno real del usuario porque requiere de la definición de una tarea específica sobre la cual calcularla, y además se necesita que los usuarios respondan un cuestionario de satisfacción, por lo cual no es una métrica transparente. Teniendo en cuenta esta limitante, el esfuerzo de interacción podría usarse por ejemplo en reemplazo de SUM para evaluar la usabilidad de una página web sin depender de una tarea y sin que los usuarios sepan que están siendo evaluados.

El hecho que no se haya encontrado una relación entre el esfuerzo y el tiempo estimado por un modelo de simulación como KLM, puede deberse a diferentes factores. Uno de ellos es que el tiempo promedio por widget utilizado en la comparación, puede no ser una forma

precisa de normalizar el tiempo dado que no considera que los distintos tipos de widgets tienen diferentes complejidades. Por dar un ejemplo, probablemente no sea lo mismo en términos de tiempo ingresar el nombre del usuario en un campo de texto que seleccionar un valor en un menú desplegable que tiene al menos 10 opciones. Más allá de la forma de normalizar el tiempo y que la estimación del tiempo total provista por la herramienta utilizada puede ser imprecisa, también puede ocurrir que no se observe una relación entre el tiempo estimado y el esfuerzo, simplemente porque el esfuerzo de interacción es una métrica que tiene en cuenta otras cuestiones además del tiempo de la interacción. Lo cierto es que para analizar estos factores se requieren nuevos experimentos que estudien de manera exhaustiva la posible correlación entre el esfuerzo y el tiempo estimado por KLM.

La principal limitación del experimento tiene que ver con que los valores de esfuerzo de las páginas analizadas fueron similares, siendo todos ellos cercanos a 1. Esto puede explicarse a partir de 2 cuestiones: el diseño de las tareas y los usuarios reclutados. En cuanto al diseño de las tareas, si bien se utilizaron páginas web reales que incluyen actividades que los usuarios realizan a diario, lo cierto es que por cuestiones técnicas las tareas tuvieron que reducirse a la interacción con una sola página web, lo cual hizo que en algunos casos estas tareas se realicen fuera de su contexto habitual. Un ejemplo concreto de este contexto podría ser que el usuario tenga que comprar un pasaje antes de poder hacer el Check-in para un vuelo. Posiblemente, el realizar la tarea en su contexto real le agregue una mayor complejidad y esto se refleje en un mayor esfuerzo de interacción.

Con respecto a los usuarios, para que el esfuerzo de interacción resultante sea representativo se busco seleccionar una muestra heterogénea de voluntarios, con diversas formaciones y experiencias previas utilizando la web. Sin embargo, siguiendo la misma línea de lo que se comentó en la discusión del capítulo 5, es importante mencionar que no fue sencillo reclutar una gran cantidad de usuarios, en particular fue difícil llegar a aquellos que podrían experimentar un mayor esfuerzo de interacción.

Más allá de las limitaciones anteriores, las pruebas realizadas sirvieron para una validación inicial del esfuerzo de interacción. Queda pendiente realizar un nuevo experimento incluyendo una mayor cantidad de páginas web y de usuarios para analizar la relación del esfuerzo con SUM y KLM, así como también comparar el esfuerzo de interacción con otras métricas que contribuyan a la evaluación de la experiencia de usuario.

Capítulo 8

Conclusiones y trabajo futuro

En este trabajo se presentó un conjunto de métodos y herramientas que pretenden dar soporte en las tareas de mejora de la experiencia de usuario en aplicaciones web. Partiendo de una interfaz de usuario web en la que los usuarios experimentan ciertos problemas de interacción, los diseñadores de UX pueden utilizar UX-Painter para explorar posibles soluciones en el mismo navegador y sin tener que codificar los cambios manualmente. La ventaja principal de este mecanismo es que las versiones generadas con UX-Painter no alteran la funcionalidad de la aplicación, por lo cual el equipo de UX puede realizar pruebas con usuarios para determinar la versión que mejor se desempeña en un contexto específico. Al momento de realizar estas pruebas, el equipo de UX también tiene a disposición UX-Analyzer, una herramienta para evaluar y comparar versiones alternativas de una página web en términos del esfuerzo de interacción realizado por los usuarios. Esta métrica de esfuerzo se calcula automáticamente a partir de modelos de predicción que se alimentan de micro-medidas que se obtienen analizando los eventos de interacción generados por los usuarios. Al tratarse de una métrica transparente, el esfuerzo de interacción da la posibilidad de evaluar una UI con múltiples usuarios.

A continuación, se realiza un repaso de la contribución detallada en el capítulo introductorio junto con un breve resumen del trabajo realizado.

8.1. Contribuciones

Un método de EUD para explorar variantes de diseño utilizando la técnica de refactoring.

La propuesta de generar alternativas de diseño mediante la aplicación asistida de CSWRs fue validada mediante entrevistas con diseñadores de UX que trabajan en la industria. En estas entrevistas, los participantes destacaron la posibilidad de modificar aspectos de la interacción del usuario sin requerir asistencia de los desarrolladores, y la rapidez con la que se pueden generar estos cambios en el diseño.

La principal limitación de este enfoque es que solo se pueden realizar aquellos cambios de diseño que estén contemplados por alguno de los CSWRs del catálogo. En particular, los diseñadores entrevistados mencionaron que los CSWRs provistos son apropiados para interfaces que contienen formularios, pero pueden ser limitados para otros casos. Si bien esta limitante es razonable, lo cierto es que no tiene que ver con el uso en sí de los CSWRs sino con el catálogo provisto. En este sentido, en el futuro habrá que seguir trabajando en el desarrollo de nuevos refactorings para darle mayor libertad a los diseñadores al momento de explorar variantes de diseño.

La incorporación de nuevos refactorings al catálogo desarrollado en trabajos anteriores.

Los nuevos refactorings que se agregaron al catálogo son: *Format Input*, *Turn Input into Select*, *Turn Input into Textarea* y *Turn Select into Autocomplete*. Además del desarrollo de estos nuevos CSWRs, gracias al enfoque de EUD también se incorporaron los refactorings que antes eran sugeridos por no poder automatizarse, estos son *Distribute Menu* y *Split Page*.

Un mecanismo de adaptación de estilos que permite personalizar ciertas propiedades estéticas de los cambios de diseño impuestos por los refactorings.

Este mecanismo no solamente fue desarrollado como una mejora a los refactorings para minimizar las alteraciones que puede generar la modificación de una página web en el navegador, sino que también tiene el propósito de darle al diseñador más opciones al momento de aplicar un refactoring. A los diseñadores entrevistados les pareció apropiada esta idea, pero también sugirieron algunas mejoras como por ejemplo agregar la posibilidad de editar los valores de las propiedades CSS.

Una herramienta denominada UX-Painter que implementa los 3 puntos anteriores.

Más allá del método de EUD, los diseñadores entrevistados también observaron el funcionamiento de UX-Painter y reflexionaron sobre el mismo. Además de destacar la facilidad de uso de la herramienta, mencionaron algunos cambios que se podrían realizar que tienen que ver con simplificar el proceso de selección y aplicación de un refactoring, teniendo en cuenta que en un futuro el catálogo de CSWRs podría crecer considerablemente.

Con respecto a la funcionalidad de UX-Painter de generar una implementación preliminar de los refactorings aplicados en una versión, si bien los primeros resultados mostraron que puede ser de gran ayuda para los desarrolladores, lo cierto es que aún está en una etapa temprana de implementación. Por lo cual, se requiere continuar con su desarrollo y realizar pruebas para determinar su utilidad.

Una métrica llamada *esfuerzo de interacción* que sirve para evaluar el funcionamiento de los distintos elementos o widgets de la UI que se ven modificados por los refactorings.

El esfuerzo de interacción es básicamente un puntaje unificado que permite comparar el funcionamiento de widgets alternativos que son útiles para una misma tarea. Este puntaje que va de 1 a 4 lo asigna un experto en UX en función de su análisis subjetivo de la interacción del usuario. La ventaja adicional de contar con una métrica de este tipo es que al evaluar individualmente los elementos de una UI, es de gran ayuda para localizar los problemas de interacción que pueden experimentar los usuarios. Además, los widgets son componentes estándar que se utilizan en todas las páginas web, por lo cual la métrica es útil para cualquier aplicación sin importar su tipo, dominio, etc.

Para validar el esfuerzo de interacción, se lo comparó con otros modelos y métricas ya establecidos en el campo de evaluación de la UX. Los resultados de las páginas web analizadas mostraron que el esfuerzo por un lado no tiene una relación con el tiempo de tarea estimado por el modelo KLM, y por el otro que es inverso a la usabilidad (de acuerdo a la métrica SUM), es decir, a mayor esfuerzo menor usabilidad y viceversa. Teniendo en cuenta que estas pruebas se realizaron en unos pocos sitios web y con una cantidad de usuarios relativamente pequeña, será necesario repetir el experimento a mayor escala para determinar si estos resultados obtenidos son generalizables.

Un conjunto de *micro-medidas* para seis tipos de widget que cuantifican diferentes aspectos de la interacción del usuario.

Teniendo en cuenta que al esfuerzo de interacción lo asignan expertos en UX, estas micro-medidas fueron desarrolladas con el propósito de determinar si era posible predecir el esfuerzo en 6 tipos de widget que comúnmente se ven modificados por los refactorings, a partir del análisis automático de la interacción de los usuarios. Para desarollar las micro-medidas, se realizó un relevamiento de la bibliografía y se realizaron pruebas de usuario en las que participaron expertos en UX.

Más allá de la utilidad de las micro-medidas para calcular el esfuerzo de interacción, también son un aporte en sí mismas porque permiten analizar distintos aspectos de la interacción de los usuario, como movimientos de mouse y eventos de teclado, al nivel de los elementos que componen una página web.

Modelos de predicción que permiten estimar el esfuerzo de interacción de un widget utilizando como entrada las micro-medidas desarrolladas.

Una vez elaboradas las micro-medidas, se desarrollaron árboles de decisión para evaluar la predicción del esfuerzo de interacción en los 6 tipos de widget analizados. Los modelos resultantes confirmaron que es posible estimar el esfuerzo a partir de las micro-medidas, dado que un MAE=0.27 se considera aceptable para esta tarea de predicción. El hecho de utilizar modelos de caja blanca como los árboles de decisión, permitió interpretar y analizar la importancia que cobra cada micro-medida en el proceso de predicción, lo cual puede ser de gran utilidad para un futuro refinamiento de los modelos desarrollados.

Una herramienta llamada UX-Analyzer que utiliza los modelos de predicción para calcular y visualizar el esfuerzo de interacción de distintas versiones de una página web.

Esta herramienta da la posibilidad de monitorear el esfuerzo de interacción de diferentes páginas web. La principal ventaja que proporciona es que el cálculo del esfuerzo es completamente transparente para los usuarios, por lo cual la herramienta es apropiada para comparar diseños alternativos, ya sea realizando algunas pruebas reclutando unos pocos usuarios o incluyendo a los usuarios reales de aplicación mediante experimentos online.

8.2. Trabajos futuros

A partir del trabajo presentado han surgido los siguientes trabajos futuros que se enumeran a continuación:

- **Extender el catálogo de CSWRs.** Si bien en este trabajo se incorporaron nuevos refactorings al catálogo, siempre se pueden agregar nuevas transformaciones para darle más opciones a los diseñadores al momento de generar alternativas de diseño. En particular, gran parte de los CSWRs contemplados actualmente son apropiados para formularios, por lo cual sería importante contar también con CSWRs que sirvan para otros casos, por ejemplo para aquellas páginas web en las que usuario más que ingresar datos consume contenido. Esto requiere del estudio de patrones de interacción en este tipo de aplicaciones web.

  Por otro lado, además de la incorporación de nuevos refactorings, el método propuesto en este trabajo para aplicar los CSWRs en el que los diseñadores son guiados paso a paso por una herramienta, también da la posibilidad de mejorar los refactorings que ya existen, por ejemplo agregando nuevas configuraciones o parámetros que le den al diseñador un mayor control de los cambios realizados.

- **Realizar mejoras a UX-Painter.** Este punto está relacionado con realizar cambios en la herramienta que mejoren su experiencia de uso en general. Alguna de estas mejoras fueron mencionadas por los diseñadores entrevistados, como por ejemplo la de simplificar el proceso de aplicación de un refactoring.

- **Validar UX-Painter en un ciclo de desarrollo ágil.** Las evaluaciones realizadas permitieron por un lado validar el enfoque propuesto mediante entrevistas que diseñadores que trabajan en la industria, y por el otro analizar la usabilidad de UX-Painter. Sin embargo, es necesario evaluar la herramienta en un contexto real de desarrollo para determinar su utilidad, tanto para los diseñadores como para los desarrolladores.

- **Continuar con la generación de código de los refactorings.** Con el objetivo de facilitarle el trabajo a los desarrolladores al momento de implementar los refactorings, se propuso la generación automática de una posible implementación para cada uno que pueda utilizarse como base para el desarrollo. Esto se trató de una prueba de concepto, para la cual se utilizó ReactJS, una librería muy usada para construir interfaces web. Por lo cual, queda pendiente validar con desarrolladores la utilidad de estas soluciones

generadas. Comprobada la validez de esta propuesta, habrá que considerar también otros frameworks y librerías utilizados para desarrollar páginas web.

- **Extender la métrica del esfuerzo de interacción.** Considerando que el esfuerzo se calcula automáticamente en 6 tipos de widget que son propios de formularios web, sería importante poder calcularlo también sobre otro tipo de elementos que permitan evaluar páginas web que no contengan formularios. Para esto, será necesario realizar un estudio de los diferentes tipos de interfaces web que existen y de los elementos interactivos que más se utilizan en éstas.

- **Incorporar otro tipo de métricas que también contribuyan a la evaluación de la UX.** Aunque en este trabajo la evaluación de páginas web se centra en el esfuerzo de interacción, el objetivo a largo plazo es contar con un conjunto de métricas que evalúen distintos aspectos de la experiencia de usuario. En ese sentido, en un futuro podrían agregarse nuevas métricas para evaluar el impacto de los CSWRs, por ejemplo aquellas que están más relacionadas con las cuestiones hedónicas de la UX, como es el caso de las que evalúan propiedades estéticas.

- **Validar UX-Analyzer.** Esta herramienta se trata de una prueba de concepto para visualizar el esfuerzo de interacción. De manera que será necesario evaluar la herramienta con diseñadores para validar el soporte que proporciona en cuanto a la evaluación y comparación de diseños alternativos en páginas web.

8.3. Publicaciones científicas

Julián Grigera, Juan Cruz Gardey, Alejandra Garrido, and Gustavo Rossi. (2018). **Live versioning of web applications through refactoring.** In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (pp. 872-875).

Julián Grigera, Juan Cruz Gardey, Andrés Rodriguez, Alejandra Garrido, and Gustavo Rossi. 2019. **One Metric for All: Calculating Interaction Effort of Individual Widgets.** In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (CHI EA '19).

Juan Cruz Gardey and Alejandra Garrido. 2020. **User Experience Evaluation through Automatic A/B Testing.** In Proceedings of the 25th International Conference on Intelligent User Interfaces Companion (IUI '20). Association for Computing Machinery, New York, NY, USA, 25–26.

Juan Cruz Gardey, Alejandra Garrido, Sergio Firmenich, Julián Grigera, and Gustavo Rossi. 2020. **UX-Painter: An Approach to Explore Interaction Fixes in the Browser.** Proceedings of the ACM on Human-Computer Interaction. 4, EICS, Article 89 (June 2020), 21 pages.

Juan Cruz Gardey, Julian Grigera, Andres Rodríguez, Gustavo Rossi, and Alejandra Garrido. (2022). **Predicting interaction effort in web interface widgets.** International Journal of Human-Computer Studies, 168, 102919.

Juan Cruz Gardey, Julian Grigera, Andres Rodríguez, Gustavo Rossi, and Alejandra Garrido. (2022). **An Interaction Effort Score for Web Pages.** In Proceedings of the 18th International Conference on Web Information Systems and Technologies - WEBIST (pp. 439-443).

Juan Cruz Gardey, Julian Grigera, Gustavo Rossi, and Alejandra Garrido. (2023). **UX-Painter: Fostering UX Improvement in an Agile Setting.** In Agile Methods. WBMA 2021. Communications in Computer and Information Science, vol 1642. Springer, Cham.

Acrónimos

- **UX:** User Experience (Experiencia de Usuario).
- **DCU:** Diseño Centrado en el Usuario.
- **UI:** User interface (Interfaz de Usuario).
- **CSWR:** Client-side web refactoring.
- **EUD:** End-user Development.
- **HCI:** Human-computer Interaction.
- **DOM:** Document Object Model.
- **HTML:** HyperText Markup Language.
- **KLM:** Keystroke Level Model.
- **AI:** Artificial Intelligence (Inteligencia Artificial).
- **URL:** Unified Resource Locator.
- **CSS:** Cascading Style Sheets.
- **API:** Application Programming Interface.
- **MAE:** Mean Absolute Error (Error Absoluto Medio).
- **SPA:** Single Page Application.
- **SUS:** Single Usability Scale.
- **SUM:** Single Usability Metric.

Glosario

Widget

Elemento de una página web que le permite al usuario interactuar con la misma. Si bien es un concepto amplio, en el contexto de este trabajo se usa para referirse a campos de formularios, links y botones.

Campo de texto

Tipo de widget nativo de HTML que permite ingresar texto en un formulario. Corresponde al tag `<input type="text"/>`.

Menú desplegable

Tipo de widget nativo que al hacerle click despliega una lista de opciones para elegir una. Corresponde al tag `<select>...</select>`.

Link

Aunque el concepto de link se refiere estrictamente al elemento que permite navegar de una página a otra (tag `<a></a>`), en este trabajo se usa el concepto de link para referirse también a los botones (`<button>`). Esto se debe a que en términos de interacción son muy similares: elementos clickeables que provocan cambios en una página web.

Botones de radio

Tipo de widget nativo que muestra una lista de opciones en la que cada una posee una casilla que permite seleccionarla. Corresponde a una colección de `<input type="radio"/>`.

Menú desplegable de fecha

Tipo de widget muy utilizado que consiste en 3 menús desplegables contiguos para seleccionar el día, mes y año de una fecha.

Datepicker

Tipo de widget que despliega un calendario para que el usuario pueda seleccionar una fecha. En este calendario se puede navegar por los diferentes meses y años.

Bibliografía

[Aldalur et al., 2017] Aldalur, I., Winckler, M., Díaz, O., and Palanque, P. (2017). *Web Augmentation as a Promising Technology for End User Development*, pages 433–459. Springer International Publishing.

[Ambler and Sadalage, 2006] Ambler, S. W. and Sadalage, P. J. (2006). *Refactoring Databases: Evolutionary Database Design.* Addison-Wesley Professional.

[Arroyo et al., 2006] Arroyo, E., Selker, T., and Wei, W. (2006). Usability tool for analysis of web designs using mouse tracks. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 484–489.

[Atterer et al., 2006] Atterer, R., Wnuk, M., and Schmidt, A. (2006). Knowing the user's every move: User activity tracking for website usability evaluation and implicit interaction. In *Proceedings of the 15th International Conference on World Wide Web*, pages 203–212.

[Attig et al., 2019] Attig, C., Then, E., and Krems, J. F. (2019). Show me how you click, and i'll tell you what you can: predicting user competence and performance by mouse interaction parameters. In *Advances in Intelligent Systems and Computing*, pages 801–806. Springer, Cham.

[Baddeley, 1979] Baddeley, A. D. (1979). *The Psychology of Memory.* Basic Books.

[Bailly et al., 2014] Bailly, G., Oulasvirta, A., Brumby, D. P., and Howes, A. (2014). Model of visual search and selection time in linear menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3865–3874. ACM.

[Bakaev et al., 2017] Bakaev, M., Mamysheva, T., and Gaedke, M. (2017). Current trends in automating usability evaluation of websites: Can you manage what you can't measure?

In *Proceedings - 2016 11th International Forum on Strategic Technology, IFOST 2016*, pages 510–514.

[Beck, 2000] Beck, K. (2000). *Extreme programming explained: embrace change*. Addison-Wesley Professional.

[Beltramelli, 2018] Beltramelli, T. (2018). pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 1–6. ACM.

[Bosetti et al., 2022] Bosetti, G., Tacuri, A., Gambo, I., Firmenich, S., Rossi, G., Winckler, M., and Fernandez, A. (2022). Andes: An approach to embed search services on the web browser. *Computer Standards & Interfaces*, 82:103633.

[Breiman et al., 1984] Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. (1984). *Classification and Regression Trees*. Chapman and Hall/CRC.

[Brhel et al., 2015] Brhel, M., Meth, H., Maedche, A., and Werder, K. (2015). Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology*, 61:163–181.

[Brooke, 1996] Brooke, J. (1996). Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189:4–7.

[Budiu, 2013] Budiu, R. (2013). Interaction cost. https://www.nngroup.com/articles/interaction-cost-definition/. Accedido el 31/01/2023.

[Burzacca and Paternò, 2013] Burzacca, P. and Paternò, F. (2013). Remote usability evaluation of mobile web applications. In Kurosu, M., editor, *Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments*, pages 241–248. Springer Berlin Heidelberg.

[Buschek et al., 2020] Buschek, D., Anlauff, C., and Lachner, F. (2020). Paper2wire a case study of user-centred development of machine learning tools for ux designers. In *Proceedings of the Conference on Mensch und Computer*, pages 33–41. ACM.

[Cabot and Gómez, 2008] Cabot, J. and Gómez, C. (2008). A catalogue of refactorings for navigation models. In *2008 Eighth International Conference on Web Engineering*, pages 75–85. IEEE.

[Card et al., 1980] Card, S. K., Moran, T. P., and Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23:396–410.

[Carta et al., 2011] Carta, T., Paternò, F., and de Santana, V. F. (2011). Web usability probe: A tool for supporting remote usability evaluation of web sites. In Campos, P., Graham, N., Jorge, J., Nunes, N., Palanque, P., and Winckler, M., editors, *Human-Computer Interaction – INTERACT 2011*, pages 349–357. Springer Berlin Heidelberg.

[Chen et al., 2012] Chen, F., Ruiz, N., Choi, E., Epps, J., Khawaja, M. A., Taib, R., Yin, B., and Wang, Y. (2012). Multimodal behavior and interaction as indicators of cognitive load. *ACM Transactions on Interactive Intelligent Systems*, 2:1–36.

[Chen et al., 2016] Chen, F., Zhou, J., Wang, Y., Yu, K., Arshad, S. Z., Khawaji, A., and Conway, D. (2016). *Theoretical Aspects of Multimodal Cognitive Load Measures*, pages 33–71. Springer International Publishing.

[Cunningham, 1992] Cunningham, W. (1992). The wycash portfolio management system. In *Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum) - OOPSLA '92*, pages 29–30. ACM Press.

[de Santana and Baranauskas, 2015] de Santana, V. F. and Baranauskas, M. C. C. (2015). Welfit: A remote evaluation tool for identifying web usage patterns through client-side logging. *International Journal of Human-Computer Studies*, 76:40–49.

[DeLeeuw and Mayer, 2008] DeLeeuw, K. E. and Mayer, R. E. (2008). A comparison of three measures of cognitive load: Evidence for separable measures of intrinsic, extraneous, and germane load. *Journal of Educational Psychology*, 100:223–234.

[Dias et al., 2019] Dias, M. C., Cepeda, C., Rindlisbacher, D., Battegay, E., Cheetham, M., and Gamboa, H. (2019). Predicting response uncertainty in online surveys: A proof of concept. *BIOSIGNALS 2019 - 12th International Conference on Bio-Inspired Systems and Signal Processing, Proceedings; Part of 12th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2019*, pages 155–162.

[Dingli and Mifsud, 2011] Dingli, A. and Mifsud, J. (2011). Useful: A framework to mainstream web site usability through automated evaluation. *International Journal of Human Computer Interaction (IJHCI)*.

[Dou et al., 2019] Dou, Q., Zheng, X. S., Sun, T., and Heng, P. A. (2019). Webthetics: Quantifying webpage aesthetics with deep learning. *International Journal of Human Computer Studies*, 124:56–66.

[Díaz et al., 2016] Díaz, O., Aldalur, I., Arellano, C., Medina, H., and Firmenich, S. (2016). Web mashups with webmakeup. In Daniel, F. and Pautasso, C., editors, *Rapid Mashup Development Tools*, volume 591, pages 82–97. Springer International Publishing.

[Díaz et al., 2013] Díaz, O., Arellano, C., and Azanza, M. (2013). A language for end-user web augmentation. *ACM Transactions on the Web*, 7:1–51.

[Fernandez et al., 2011] Fernandez, A., Insfran, E., and Abrahão, S. (2011). Usability evaluation methods for the web: A systematic mapping study. *Information and Software Technology*, 53:789–817.

[Fitts, 1954] Fitts, P. M. (1954). The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47:381–391.

[Fowler et al., 1999] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

[Gajos et al., 2010] Gajos, K. Z., Weld, D. S., and Wobbrock, J. O. (2010). Automatically generating personalized user interfaces with supple. *Artificial Intelligence*, 174:910–950.

[Garcia et al., 2019] Garcia, A., da Silva, T. S., and Silveira, M. (2019). Artifact-facilitated communication in agile user-centered design. In *Agile Processes in Software Engineering and Extreme Programming*, volume 2, page 260. Springer International Publishing.

[Gardey. et al., 2022] Gardey., J., Grigera., J., Rodríguez., A., Rossi., G., and Garrido., A. (2022). An interaction effort score for web pages. In *Proceedings of the 18th International Conference on Web Information Systems and Technologies - WEBIST,*, pages 439–443. SciTePress.

[Gardey et al., 2020] Gardey, J. C., Garrido, A., Firmenich, S., Grigera, J., and Rossi, G. (2020). Ux-painter: An approach to explore interaction fixes in the browser. *Proceedings of the ACM on Human-Computer Interaction*, 4.

[Gardey et al., 2022] Gardey, J. C., Grigera, J., Rodríguez, A., Rossi, G., and Garrido, A. (2022). Predicting interaction effort in web interface widgets. *International Journal of Human-Computer Studies*, 168:102919.

[Gardey et al., 2023] Gardey, J. C., Grigera, J., Rossi, G., and Garrido, A. (2023). Ux-painter: Fostering ux improvement in an agile setting. In Rocha, C., Júnior, C. S., Fernando, D. S., and da Silva, T. S., editors, *Agile Methods, WBMA 2021*, pages 54–65. Springer International Publishing.

[Garrett, 2002] Garrett, J. J. (2002). *The Elements of User Experience: User-Centered Design for the Web*. Peachpit Pr.

[Garrido et al., 2017] Garrido, A., Firmenich, S., Grigera, J., Rossi, G., and Ieee (2017). Data-driven usability refactoring: Tools and challenges. *6th International Workshop on Software Mining (Softwaremining)*, pages 52–55.

[Garrido et al., 2013] Garrido, A., Firmenich, S., Rossi, G., Grigera, J., Medina-Medina, N., and Harari, I. (2013). Personalized web accessibility using client-side refactoring. *IEEE Internet Computing*, 17:58–66.

[Garrido et al., 2007] Garrido, A., Rossi, G., and Distante, D. (2007). Model refactoring in web applications. In *2007 9th IEEE International Workshop on Web Site Evolution*, pages 89–96. IEEE.

[Garrido et al., 2011] Garrido, A., Rossi, G., and Distante, D. (2011). Refactoring for usability in web applications. *IEEE Software*, 28:60–67.

[Ghiani et al., 2016] Ghiani, G., Paternò, F., Spano, L. D., and Pintori, G. (2016). An environment for end-user development of web mashups. *International Journal of Human-Computer Studies*, 87:38–64.

[Grigera, 2017] Grigera, J. (2017). *Self-Refactoring: mejoras automáticas de usabilidad para aplicaciones web*. PhD thesis, Facultad de Informática, Universidad Nacional de La Plata, Argentina.

[Grigera et al., 2018] Grigera, J., Gardey, J. C., Garrido, A., and Rossi, G. (2018). Live versioning of web applications through refactoring. In *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*.

[Grigera et al., 2016] Grigera, J., Garrido, A., Panach, J. I., Distante, D., and Rossi, G. (2016). Assessing refactorings for usability in e-commerce applications. *Empirical Software Engineering*, pages 1224–1271.

[Grigera et al., 2017a] Grigera, J., Garrido, A., Rivero, J. M., and Rossi, G. (2017a). Automatic detection of usability smells in web applications. *International Journal of Human Computer Studies*, 97:129–148.

[Grigera et al., 2017b] Grigera, J., Garrido, A., and Rossi, G. (2017b). Kobold: Web usability as a service. *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, pages 990–995.

[Grigera et al., 2019] Grigera, J., Rodriguez, A., Gardey, J., Garrido, A., and Rossi, G. (2019). One metric for all: Calculating interaction effort of individual widgets. In *Conference on Human Factors in Computing Systems - Proceedings*.

[Gütl et al., 2005] Gütl, C., Pivec, M., Trummer, C., García-Barrios, V. M., Mödritscher, F., Pripfl, J., and Umgeher, M. (2005). Adele (adaptive e-learning with eye-tracking): Theoretical background, system architecture and application scenarios. *European Journal of Open, Distance and E-Learning (EURODL)*, 8.

[Hassenzahl et al., 2001] Hassenzahl, M., Beu, A., and Burmester, M. (2001). Engineering joy. *IEEE Software*, 18:70–76.

[Hassenzahl and Tractinsky, 2006] Hassenzahl, M. and Tractinsky, N. (2006). User experience - a research agenda. *Behaviour & Information Technology*, 25:91–97.

[Hollender et al., 2010] Hollender, N., Hofmann, C., Deneke, M., and Schmitz, B. (2010). Integrating cognitive load theory and concepts of human-computer interaction. *Computers in Human Behavior*, 26:1278–1288.

[Horwitz et al., 2017] Horwitz, R., Kreuter, F., and Conrad, F. (2017). Using mouse movements to predict web survey response difficulty. *Social Science Computer Review*, 35:388–405.

[Hurst et al., 2007] Hurst, A., Hudson, S. E., and Mankoff, J. (2007). Dynamic detection of novice vs. skilled use without a task model. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–280.

[ISO, 2018] ISO (2018). *ISO 9241-11:2018 - Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. ISO/TC 159/SC 4.

[ISO, 2019] ISO (2019). *ISO 9241-210:2019 - Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*. ISO/TC 159/SC 4.

[Ivory and Hearst, 2001] Ivory, M. Y. and Hearst, M. A. (2001). The state of the art in automating usability evaluation of user interfaces. *ACM Computing Surveys*, 33:470–516.

[Janlert and Stolterman, 2017] Janlert, L. E. and Stolterman, E. (2017). The meaning of interactivity—some proposals for definitions and measures. *Human-Computer Interaction*, 32:103–138.

[Jurca et al., 2014] Jurca, G., Hellmann, T. D., and Maurer, F. (2014). Integrating agile and user-centered design: A systematic mapping and review of evaluation and validation studies of agile-ux. In *Proceedings - 2014 Agile Conference, AGILE 2014*, pages 24–32. IEEE.

[Katsanos et al., 2013] Katsanos, C., Karousos, N., Tselios, N., Xenos, M., and Avouris, N. (2013). Klm form analyzer: Automated evaluation of web form filling tasks using human performance models. In *Human-Computer Interaction – INTERACT 2013*, pages 530–537. Springer Berlin Heidelberg.

[Ko et al., 2011] Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., and Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys*, 43:1–44.

[Kohavi and Longbotham, 2017] Kohavi, R. and Longbotham, R. (2017). Online controlled experiments and a/b testing. In *Encyclopedia of Machine Learning and Data Mining*, pages 922–929. Springer US.

[Kuusinen, 2016] Kuusinen, K. (2016). Bob: A framework for organizing within-iteration ux work in agile development. In *Integrating User-Centred Design in Agile Development*, pages 205–224. Springer International Publishing.

[Law et al., 2009] Law, E. L.-C., Roto, V., Hassenzahl, M., Vermeeren, A. P., and Kort, J. (2009). Understanding, scoping and defining user experience. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 719–728. ACM.

[Leino et al., 2019] Leino, K., Oulasvirta, A., and Kurimo, M. (2019). Rl-klm automating keystroke-level modeling with reinforcement learning. In *Proceedings of the 24th International Conference on Intelligent User Interfaces*, pages 476–480. ACM.

[Lewis et al., 2013] Lewis, J. R., Utesch, B. S., and Maher, D. E. (2013). Umux-lite: when there's no time for the sus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2099–2102. ACM.

[Lewis et al., 2015] Lewis, J. R., Utesch, B. S., and Maher, D. E. (2015). Investigating the correspondence between umux-lite and sus scores. In *Design, User Experience, and Usability: Design Discourse*, pages 204–211. Springer International Publishing.

[Li et al., 2018] Li, Y., Bengio, S., and Bailly, G. (2018). Predicting human performance in vertical menu selection using deep learning. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 1–7.

[Lieberman et al., 2006] Lieberman, H., Paternò, F., Klann, M., and Wulf, V. (2006). End-user development: An emerging paradigm. In *Human-Computer Interaction Series*. Springer Netherlands, Dordrecht.

[Lim et al., 2016] Lim, W. T., Wang, L., Wang, Y., and Chang, Q. (2016). Housing price prediction using neural networks. In *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 518–522. IEEE.

[McNally et al., 2018] McNally, S., Roche, J., and Caton, S. (2018). Predicting the price of bitcoin using machine learning. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 339–343. IEEE.

[Michailidou et al., 2021] Michailidou, E., Eraslan, S., Yesilada, Y., and Harper, S. (2021). Automated prediction of visual complexity of web pages: Tools and evaluations. *International Journal of Human-Computer Studies*, 145.

[Michailidou et al., 2008] Michailidou, E., Harper, S., and Bechhofer, S. (2008). Visual complexity and aesthetic perception of web pages. In *Proceedings of the 26th annual ACM international conference on Design of communication*, pages 215–224. ACM.

[Moran et al., 2020] Moran, K., Bernal-Cardenas, C., Curcio, M., Bonett, R., and Poshyvanyk, D. (2020). Machine learning-based prototyping of graphical user interfaces for mobile apps. *IEEE Transactions on Software Engineering*, 46:196–221.

[Navalpakkam and Churchill, 2012] Navalpakkam, V. and Churchill, E. F. (2012). Mouse tracking: Measuring and predicting users' experience of web-based content. In *Conference on Human Factors in Computing Systems - Proceedings*, pages 2963–2972.

[Nebeling et al., 2012] Nebeling, M., Leone, S., and Norrie, M. C. (2012). Crowdsourced web engineering and design. In *International Conference on Web Engineering (ICWE)*, pages 31–45. Springer Berlin Heidelberg.

[Nielsen, 1994] Nielsen, J. (1994). 10 usability heuristics for user interface design. https://www.nngroup.com/articles/ten-usability-heuristics/. Accedido el 25/01/2023.

[Nielsen, 2005] Nielsen, J. (2005). Putting a/b testing in its place. https://www.nngroup.com/articles/putting-ab-testing-in-its-place/. Accedido el 27/01/2023.

[Opdyke, 1992] Opdyke, W. F. (1992). *Refactoring Object-Oriented Frameworks*. PhD thesis, University of Illinois at Urbana-Champaign, USA.

[Oulasvirta et al., 2018] Oulasvirta, A., Pascale, S. D., Koch, J., Langerak, T., Jokinen, J., Todi, K., Laine, M., Kristhombuge, M., Zhu, Y., Miniukovich, A., Palmas, G., and Weinkauf, T. (2018). Aalto interface metrics (aim): A service and codebase for computational gui evaluation. In *31st Annual ACM Symposium on User Interface Software and Technology*, pages 16–19. Association for Computing Machinery, Inc.

[Paternò et al., 2017] Paternò, F., Schiavone, A. G., and Conti, A. (2017). Customizable automatic detection of bad usability smells in mobile accessed web applications. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–11. Association for Computing Machinery.

[Reinecke et al., 2013] Reinecke, K., Yeh, T., Miratrix, L., Mardiko, R., Zhao, Y., Liu, J., and Gajos, K. Z. (2013). Predicting users' first impressions of website aesthetics with a quantification of perceived visual complexity and colorfulness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2049–2058. ACM.

[Rubin and Chisnell, 2008] Rubin, J. and Chisnell, D. (2008). *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*. Willey, 2nd edition edition.

[Rubin, 2012] Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional.

[Salminen et al., 2019] Salminen, J., Sengün, S., gyo Jung, S., and Jansen, B. J. (2019). Design issues in automatically generated persona profiles. In *Proceedings of the 2019 Conference on Human Information Interaction and Retrieval*, pages 225–229. ACM.

[Sauro and Kindlund, 2005a] Sauro, J. and Kindlund, E. (2005a). A method to standardize usability metrics into a single score. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 401–409. ACM.

[Sauro and Kindlund, 2005b] Sauro, J. and Kindlund, E. (2005b). Using a single usability metric (sum) to compare the usability of competing products. In *Proceeding of the Human Computer Interaction International Conference (HCII 2005)*.

[Silva et al., 2011] Silva, T. S. D., Martin, A., Maurer, F., and Silveira, M. (2011). User-centered design and agile methods: A systematic review. In *Agile Conference*, pages 77–86. IEEE.

[Silva et al., 2018] Silva, T. S. D., Silveira, M. S., Maurer, F., and Silveira, F. F. (2018). The evolution of agile uxd. *Information and Software Technology*, 102:1–5.

[Silva et al., 2013] Silva, T. S. D., Silveira, M. S., Melo, C. D. O., and Parzianello, L. C. (2013). Understanding the ux designer's role within agile teams. In Marcus, A., editor, *Design, User Experience, and Usability. Design Philosophy, Methods, and Tools*, pages 599–609. Springer.

[Soukoreff and MacKenzie, 2004] Soukoreff, R. W. and MacKenzie, I. S. (2004). Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci. *International Journal of Human-Computer Studies*, 61:751–789.

[Speicher et al., 2014] Speicher, M., Both, A., and Gaedke, M. (2014). Ensuring web interface quality through usability-based split testing. *Icwe, LNCS 8541*, pages 93–110.

[Speicher et al., 2015] Speicher, M., Both, A., and Gaedke, M. (2015). S.o.s. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1005–1014. ACM.

[Suleri et al., 2019] Suleri, S., Pandian, V. P. S., Shishkovets, S., and Jarke, M. (2019). Eve: A sketch-based software prototyping workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–6. ACM.

[Todi et al., 2016] Todi, K., Weir, D., and Oulasvirta, A. (2016). Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pages 543–555. ACM.

[Vermeeren et al., 2010] Vermeeren, A. P. O. S., Law, E. L.-C., Roto, V., Obrist, M., Hoonhout, J., and Väänänen-Vainio-Mattila, K. (2010). User experience evaluation methods. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, pages 521–530. ACM.

[Wong and Hong, 2007] Wong, J. and Hong, J. (2007). Making mashups with marmite: Towards end-user programming for the web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1435–1444. Association for Computing Machinery.

[Yang et al., 2020] Yang, B., Wei, L., and Pu, Z. (2020). Measuring and improving user experience through artificial intelligence-aided design. *Frontiers in Psychology*, 11.

[Zhai et al., 2016] Zhai, Z., Cheng, B., Wang, Z., Liu, X., Liu, M., and Chen, J. (2016). Design and implementation. In *Proceedings of the 25th International Conference Companion on World Wide Web - WWW '16 Companion*, pages 143–144. ACM Press.

[Zhou et al., 2020] Zhou, J., Tang, Z., Zhao, M., Ge, X., Zhuang, F., Zhou, M., Zou, L., Yang, C., and Xiong, H. (2020). Intelligent exploration for user interface modules of mobile app with collective learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3346–3355. ACM.