UX-Painter: An Approach to Explore Interaction Fixes in the Browser

JUAN CRUZ GARDEY, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina
ALEJANDRA GARRIDO, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina
SERGIO FIRMENICH, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina
JULIÁN GRIGERA, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CICPBA & CONICET, Argentina
GUSTAVO ROSSI, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina

Usability and user interaction improvement is a central task in web development to guarantee the success of a web application. However, designers are barely able to keep up with the current development cycle because their practices are too costly, while interaction issues accumulate in applications that end-users keep suffering. In this work, we propose a method for designers to rapidly explore solutions through visual programming to the interaction problems of an application under development, even when it has been already deployed. The method is realized by a tool called UX-Painter, an exploratory tool for designers to apply quick fixes to interaction issues at the client-side of a web application without the need of any script programming knowledge. The palette of available fixes in UX-Painter are client-side web refactorings, i.e., changes to web page elements that solve specific user interaction problems without changing the underlying functionality. UX-Painter allows designers to quickly set up new versions of a web application by combining refactorings to create alternative designs for user testing or an inspection review. UX-Painter also provides the means to communicate design improvements, as a sequence of refactorings with clear semantics. We show the feedback provided by interviews with designers about UX-Painter's functionality and the results of a user test about its usability.

CCS Concepts: • **Human-centered computing → Systems and tools for interaction design**; *User interface toolkits*; • **Software and its engineering → Visual languages**.

Additional Key Words and Phrases: Web usability; interaction design; end-user programming; user experience; web refactoring.

ACM Reference Format:
Juan Cruz Gardey, Alejandra Garrido, Sergio Firmenich, Julián Grigera, and Gustavo Rossi. 2020. UX-Painter: An Approach to Explore Interaction Fixes in the Browser. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 89 (June 2020), 21 pages. https://doi.org/10.1145/3397877

1 INTRODUCTION

Usability, user interaction and user experience (which we will refer to as UX in general) are essential for web applications' success and are known to create high business value [7]. In spite of this, UX

Authors' addresses: Juan Cruz Gardey, jcgardey@lifia.info.unlp.edu.ar, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina; Alejandra Garrido, garrido@lifia.info.unlp.edu.ar, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina; Sergio Firmenich, sergio.firmenich@lifia.info.unlp.edu.ar, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina; Julián Grigera, julian.grigera@lifia.info.unlp.edu.ar, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CICPBA & CONICET, Argentina; Gustavo Rossi, gustavo@lifia.info.unlp.edu.ar, LIFIA, Fac. de Informática, Univ. Nac. de La Plata & CONICET, Argentina.

2573-0142/2020/6-ART89 $15.00
https://doi.org/10.1145/3397877

evaluation and improvement are still neglected practices, even in agile development methods which are driven by business value and customer satisfaction [3, 20]. The reason for this contradiction is the high cost of UX practices, largely due to the lack of systematic methods and artifacts to help designers keep up with the speed of agile development [7]. While there are advances in the integration of UX and agile practices, it is still hard for designers to work in sync with developers in the current product increment. Moreover, the ideal practice of agile teams working together in the same room [2] seems to be losing strength rapidly, as distributed teams are becoming popular, with UX designers usually located separately from developers. This creates another challenge in synchronizing activities and communicating design decisions among stakeholders [7].

The study of Garcia et al. [13] on the artifacts used for communication between designers and developers shows a concentration of artifacts for requirements' clarification and up-front design (user stories, wireframes, prototypes, personas), while evidencing a lack of artifacts when UX improvement should take place, i.e., when solutions to existing UX problems in the current product increment must be evaluated, compared and communicated to developers to be implemented.

While there are several tools for remote user testing and event logging to help designers find UX problems from feedback of real users, there is not much help in applying fixes to those problems, or even try different alternative designs for a User Interface (UI) during and after development. Techniques such as prototypes and mockups are effective to quickly explore alternative designs, but this does not compare with applying fixes in the real context of the application being developed, to assess which of the potential solutions works better with final users. This assessment is often unfeasible due to the high cost that requires the full implementation of the potential solutions. Besides that it is a time-consuming task, UX experts may not be able to do it on their own because programming skills are needed. As a result, UX problems accumulate, and fixes which are assumed to work may actually cost a regression in UX.

To help mitigate these problems we propose a method to explore alternative designs of a web application directly in production without altering its codebase, trying out potential solutions to detected UX problems, or simply exercise some ideas for different designs in the current product increment. The exploration of alternative designs is a common practice at the start of a project and during the design and prototyping stage [3]. In fact, there are several prototyping tools intended to explore different designs for a UI that needs to be implemented. Unlike these tools that focus on designing a user interface from scratch, we propose this exploration to happen directly on the product increment that has been released and even once it has real users rising UX claims on the product. The method is realized through an End-User Development (EUD) approach, where the intended end-users are UX designers and the supported task is the rapid set up of alternative designs. The building blocks to create these alternative designs are predefined UI transformations called Client-Side Web Refactorings (CSWRs) [14]. Each CSWR is intended to solve a UX issue by changing a specific interaction directly in a web page while preserving the underlying functionality. Thus, our EUD approach allows designers to freely combine CSWRs to create new versions of a web application. These new versions may be used to perform evaluations, as inspection reviews or user tests, which may even be considered a form of A/B testing that inspects UX instead of conversion rate. Once the best version is discovered, it may be shared with developers to implement in the backend, as a series of CSWR with specifics semantics.

We have implemented an EUD tool called UX-Painter. UX-Painter is a web-extension that has a palette of 19 CSWRs that may be applied with a simple "point-and-shoot" metaphor, i.e., visually selecting the DOM elements involved and additional parameters, like style. The changes performed are saved as application versions that can be accessed at any time and exported to be re-created in others browsers. We have validated the feasibility of the exploration method with UX designers that work in the industry. Early experiments were also conducted in the context of a course about agile development incorporating User Centered Design (UCD) practices. The feedback of students in a first version of UX-Painter highlighted the need to incorporate the ability to choose between various styles for the transformed widgets. In order to add this functionality, but still keeping the tool usable for any person, we incorporated to the CSWRs the capability of generating appropriate styles for the new widgets, by capturing the styles of similar widgets in the page. Thus, a user does not need to code CSS either. Additionally, we have evaluated the usability of UX-Painter with a user test in which a subject is provided with a list of design solutions to apply.

To summarize, the proposed method and tool contribute to the process of incremental UX improvement by freely exploring interaction design refactorings, called interaction fixes for short, with the following benefits: *(i)* designers are able to apply changes to the UI of a developed web application, even in production; *(ii)* the method may be used in the context of an agile development process, to support designers in their UX improvement tasks through an EUD approach, while development proceeds; *(iii)* UX-Painter serves as a communication artifact between designers and developers, to describe design changes with specific semantics; *(iv)* the available set of interaction fixes is augmented with 6 new CSWRs: *Distribute Menu*, *Format Input*, *Split Page*, *Turn Input into Select*, *Turn Input into Textarea* and *Turn Select into Autocomplete*; *(v)* the method includes a style adaptation mechanism to prevent style disruption upon UI changes. The current palette of transformations consists of 19 CSWRs and 2 style detection mechanisms, and while it mostly applies to web forms, it is not limited to a particular type of web application regarding a specific domain or technology. Adding new refactorings to the palette is also possible.

The rest of the paper is organized as follows: Section 2 presents related work. Section 3 describes the motivation and use of our proposal in the context of agile development cycles, the requirements imposed by the proposal, the palette of CSWRs, and the style adaptation mechanism. Section 4 defines the architecture of UX-Painter and then presents a demonstration of the tool in a well-known site. Section 5 describes the different experiments that we performed to validate the tool. Section 6 presents a discussion about the experiment results and limitations of the current implementation of the tool and Section 7 shows conclusions and future work.

2 RELATED WORK

During the last 15 years, there has been extensive research in the integration of UCD with agile software development, especially since agile methods have become mainstream in software development and many of their values are in line with those of UX engineering [3, 7, 20]. In their latest survey, Da Silva et al. recognize advances in social and process integration, since agile teams usually have UX experts as members, and it has become clear that the practices of both methods must be synchronized [7]. However, we are still missing advances in the technology and artifacts dimension, and empirical results that may convince the industry to adopt them [3, 7]. Without appropriate tools and techniques to help designers work in sync with developers in the current product increment, no true integration may be accomplished.

Regarding tools for UCD practices, both research and industry have produced good results for usability evaluation and UX problem discovery. Well known tools resulting from research works include WUP [4], WELFIT [25] and W3touch [24], while industry tools include crowdsourcing platforms for user testing (e.g., uTest [19], test.io [27]) and analytics tools. While these tools may help designers in finding UX problems from feedback of real users, there is not much help in applying fixes to those problems, or even try different alternative designs for a UI during development. Designers may mock-up alternative designs to fix UX problems, but depend on the availability of developers to implement those fixes in the front-end code, before they can even test them.

Prototyping is a widely used practice in the design process of an application to generate a shared understanding between all the team members about how a UI should look like. It allows to quickly test different design ideas with the goal of reducing development costs. Also, low and high fidelity prototypes have been proposed to perform inspection evaluations and usability tests [6]. Low fidelity prototypes allow verifying the feasibility of UI interactions in terms of technical support [13]. Regarding high fidelity prototypes, modern commercial tools such as Adobe XD[1] or InVision[2] provide environments to create sophisticated designs that include animations and interactions, which give rise to perform UX evaluations in a context closer to the real application. However, there is no support to try variations in the design of real UIs to fix UX issues and evaluate them in a real context of use. We believe that evaluations in the working product are critical since unexpected UX issues not reproducible on the prototypes may arise. In this sense, our approach pursues the same goal that rapid prototyping tools that is, explore designs with minimal effort, but with the difference that we do it directly in the target application, i.e., modifying the rendered UI instead of generating a new artifact. This method would generally involve both, designers and developers, the first to propose solutions and the second to implement them in the application, with the drawback that it may produce waiting time and bottlenecks [5, 23], an extra cost that we intend to avoid.

In order to adapt an existing website quickly without requiring developers to implement them in the application, designers may use the web browser as a platform for performing UI changes. In fact, modern web browsers provide built-in tools to facilitate development and testing of an application. For instance, the browser development tools help developers and designers see, inspect, debug and change the UI on the fly, although programming knowledge is required and the changes are lost in futures visits. Moreover, web browsers also give the chance to change the appearance and behavior of web sites externally through custom web-extensions or addons. These has been used to develop different web augmentation (WA) tools that follow an EUD approach to empower end-users with environments for creating artifacts that manipulate existing and third-party web applications without depending on developers [1]. Examples of these tools are WebMakeup [8] to customize an existing web page, and those that allow to create mashups, i.e., combine information from diverse web sites into a single hub [15, 28]. Considering that Diaz [9] states that WA consists of "layering relevant content/layout/navigation over the existing Web to customize the user experience", our work can be considered a kind of WA, but with a focus on improving the user interaction without adding new functionality to the page.

Our approach is actually based on a web adaptation technique called Client-Side Web Refactoring (CSWR), which applies predefined scripts to transform web page elements through DOM manipulation, with the purpose of solving interaction issues, or bad interaction “smells”. The motivation of CSWR follows the philosophy of well-known code refactoring [11], i.e., to solve a "bad smell" through a transformation that must guarantee the preservation of the original application functionality. CSWRs were originally conceived to personalize web applications in the browser to improve accessibility [14], and later applied to improve usability [10], which has been assessed with good results [16]. With the purpose of automating the usability improvement, CSWRs have been used in a tool that reports "usability smells" by analyzing user interaction events and for each reported smell it suggests one or more CSWRs as solutions that can be applied in an assisted way [18]. Contrary to this tool that requires the target application to be used for a period of time to detect usability problems, we propose the usage of CSWRs to freely exercise different design ideas, i.e, by directly applying them to the desired UI elements, combining the WYSIWYG philosophy with the "point and shoot" metaphor. As they are predefined scripts, CSWRs can be easily injected or removed from the browser without writing code. Moreover, we have improved the CSWRs including style adaptation strategies to adjust aesthetic properties of the changes, and we have proposed new

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

CSWRs not considered before. Concerning the impact of the CSWRs in the applications, we have extended the scope to UX because the free application of CSWRs may also influence subjective aspects not included in the usability dimensions.

Regarding the EUD perspective, while most WA approaches target any type of end-users, our method is intended only for UX designers. According to Ko et al. [21], who define EUD as "programming to achieve a result of a program for personal use", the method proposed is an EUD approach to support designers in their exploration tasks. In fact, Ko mentions as an example that designers exercise EUD when they prototype a UI with a specific tool.

We could also compare UX-Painter with A/B testing tools, that is, tools that allow creating separate versions of a web application for a controlled experiment, randomizing the users that access each version [22]. While some commercial A/B testing tools like Google Optimize[3] also provide support to generate the alternative versions including WYSIWYG editors to modify the target application's UI, they are focused on aesthetics properties of a page such as colors, fonts and labels, but more complex changes require changing the source code. This is where UX-Painter surpasses A/B testing tools, providing a palette of predefined solutions to interaction problems.

Finally, we may also compare our approach with Supple, a tool intended to automatically generate an optimal UI for a specific device and user needs [12]. The Supple system is intended for end users to create a personalized UI by defining an abstract model that includes user and device constraints, which is fed to an optimization algorithm. Thus, the approach is best suited for situations where an aesthetically pleasing UI created by a human designer is less important than addressing individual abilities or devices, as it has been specifically applied to address users with motor impairments. Besides these differences, the mechanism used to generate the new design is similar to the CSWRs in the sense that it is based on the usage of alternative UI elements or widgets that serve the same purpose, ensuring that the functionality of the target UI is kept along the different designs.

3 RATIONALE BEHIND UX-PAINTER

The proposed approach aims at providing support for collaboration between developers and UX designers to improve the UX during and after the implementation of a product increment. After developing a first version of the product increment, UX-Painter allows designers and developers to work closer by using the target application itself as a design artifact to iteratively improve the UX by exploring possible solutions to problems found and implementing the one which works better.

In the context of an agile+UCD integration approach like the one proposed by Sy [26], designers work "one sprint ahead" of development creating specifications of the UIs to be implemented using prototypes or mockups. Moreover, designers must conduct UX evaluations on the product increment implemented in the preceding sprint. For this purpose they can use UX-Painter to try out alternative solutions for the UX issues found directly in the working product. Finally, once a explored design was selected, it can be added to the product backlog for its further implementation. At this point, UX-Painter serves to communicate the design changes to developers, as CSWRs are, behind doors, code snippets with clear mechanics.

3.1 Requirements

Our goal is to propose a method and associated tool to help in the integration of UX and agile practices during mid-to-late stages of development, providing support for evaluating the UX of a product increment (MVP) that has been already developed. This imposes some requirements as outlined below:

[3] https://optimize.google.com/

- *In the wild exploration.* Given that the MVP under analysis is deployed (either on a testing or production environment) and evaluations in design phase may have been already conducted, the exploration must happen directly in the UI to allow the testing in the working product with real users.
- *Application independent*: It is important that the method allows to modify the UI of any web application regardless its domain specifics or the technology used to build it.
- *No programming needed.* As we mentioned earlier, the main limitation of testing alternative solutions in the working product is that it requires time and knowledge of the programming technology used to develop the product. Applying changes to the UI without writing code saves precious time dedicated to set up the designs to be tested. This requirement poses the need of developing an EUD approach to support UX designers to create alternative designs.
- *Quick exploration.* This requirement is related with the previous one. It not only refers to the time needed to create new designs, but also to the flexibility to observe and interact with them, allowing to easily discard the changes performed without affecting the application's functionality.
- *Persistence of the designs created.* A persistence mechanism must be provided to save generated designs to allow further UX evaluations (e.g user testing, A/B testing) and to communicate the final design to be implemented to the developer team.
- *Portability.* It should be easy to re-generate the designs created in others environments, in order to allow the UX evaluations in the context of the application's users.
- *Traceability.* This requirement consists of keeping a log of the changes applied to a UI to easily communicate to developers a design that must be finally implemented. By providing to them a trace of the modifications performed together with the resulted UI, they can better understand what needs be implemented reducing communication conflicts. Each change log should provide a clear description of its purpose to facilitate the implementation.

3.2 UX-Painter palette of CSWRs

In the present work we have extended the previous catalogues of CSWRs [17] to approach UX in general, and most of all, to provide different alternatives to fix the same UX problem. This is the case, for instance, with the CSWRs *Add Datepicker*, *Date Input into Select* and *Format Input*, which may all be used to prevent errors when entering dates. Having alternative CSWRs for the same UX problem allows a designer to create alternative versions of a web application to test which solution works better. As designers have recognized in the experiments, this is an important feature of UX-Painter, since it allows a variation of A/B testing for UX to be easily applied. Although each CSWR performs a small change in the UI, major design changes can be achieved by applying a sequence of them. All CSWRs ensure that the functionality of the target application is maintained by replacing widgets with others semantically similar or simply re-ordering certain elements that already exist in the UI.

This section provides a description of each CSWR, including the UX problem that each one intends to solve. While the majority of CSWRs modify a single page element, there are some more complex ones that involve multiple elements. We next describe each of the 19 CSWRs in UX-Painter's palette.

Add Autocomplete: Provide an auto-complete feature to a text field, in order to suggest a list of expected values as the user types, making the input of popular values easier. It requires the list of the possible values as extra information.

Add Datepicker: Add a calendar to a text field to select a date in order to prevent the format errors that may arise when typing a date.

Add Form Validation: Add client-side validation of the mandatory fields in a form. The validation is triggered upon form submission. Client-side validation minimizes failed form submissions (which can be time-consuming). This CSWR requires the list of mandatory form fields.

Add Inline Validation: Add client-side validation for mandatory fields in a form, triggering the validation for each field as soon as the focus is lost. Extra information required is the list of mandatory form fields.

Add Link: Create a link in a specific section of the target page. The new link should make it easier to reach a determined page, avoiding long navigation paths. It requires the link's name and the destination URL.

Add Loading Overlay: Add an overlay when a form is submitted to make clear that the application is processing the data. The explicit feedback given by the overlay helps preventing the user's confusion caused by a frozen UI.

Add Tooltip: Add a tooltip when the user poses the mouse over an element to make its purpose clearer. The element may be a text, image, button or input. It requires the tooltip's description.

Date Input into Selects: Turn a text field for dates into three select boxes to choose day, month and year. Like *Add Datepicker*, this CSWR intends to facilitate the input of a date.

Distribute Menu: In a list of selectable items with bulk actions, add an action as an individual link next to each item, so it is easier to apply that action on a single one. This CSWR is more complex than the others since it affects multiple page elements. It requires the target list, the widgets used to select a list item, usually a checkbox, and the bulk action that will be distributed to each item in the list.

Format Input: Add a mask to a text field to limit the possible character types, in order to guide the user to the expected format. This CSWR, as well as *Add Datepicker* and *Date Input into Selects*, is intended to prevent errors than can arise when entering a value with a format constraint, but in this case also considering other formats than dates, like credit card numbers, passwords, etc. The input's format is required. For example, to limit a field to a date in the DD/MM/YYYY format the mask "00/00/0000" can be used.

Link to Top: Add a new link at the bottom of the page as the user scrolls down to go directly to the top. This link can be useful especially in long pages that need a lot of scroll. The application of this CSWR does not require any additional parameters.

Rename Element: Change the label of an interactive element (a button or a link) to make it more descriptive. Like *Add Tooltip*, this CSWR aims to improve an element's description. Static text elements can also be renamed. The new label of the element is required.

Resize Input: Change the length of a text field. By adjusting a field's length to the expected values, the users can better foresee the information that they have to fill in. This CSWR requires the new input width in pixels.

Split Page: Divide the content on a cluttered page in different sections, so it is easier to read or interact with. Only one section is shown at a time and the remaining ones are hidden. The sections can be navigated through a sections' list that is added in a specific part of the page. It requires the lists of sections in which the page is divided and the position in the page where the list with section links will be placed.

Turn Attribute into Link: Turn a static element that users intend to click like a text or an image into a link to a specific URL. It requires the destination URL.

Turn Input into Radios: Replace a text field with a limited set of common values into a radio buttons set, adding an "other" option to leave the choice of entering an unexpected option. It can be an alternative for *Add Autocomplete*. The list of common values is required.

Turn Input into Select: Replace a regular text field by a select box with a list of common values. As in the previous CSWR, this one also simplifies the choice of a given set of values, with the "other" option to enter a different one. It requires the values list to complete the select box.

Turn Input into Textarea: Replace a text field with a text area in order to extend the available space to enter a value. Like *Resize Input*, this change aims to improve the hint given to the users about the length of the expected values.

Turn Select into Autocomplete: Replace a select box with a text field that automatically suggests the possible values. It is similar to *Add Autocomplete*, but the suggested values are the options of the replaced select. This CSWR makes the selection of a specific value easier when the options list is large, since it allows users to type a value instead having to read all options in the select box to find the desired value.

3.3 Adapting widgets style

The CSWRs perform changes on a web application's DOM structure, which many times involve the addition of new widgets; an example is *Turn Input into Select*, which replaces a text field with a select box. Adding a new widget to a page may disrupt its layout, which can lead to problems such as unreadable text and cluttered navigation. Beyond the page's layout alteration, some aesthetics's properties of the new element such as the colors and fonts, may differ from the look and feel of the target page. Figure 1 illustrates an application of *Turn Input into Select*, where it can be observed that the new select box does not fit into the style of existing form's fields.

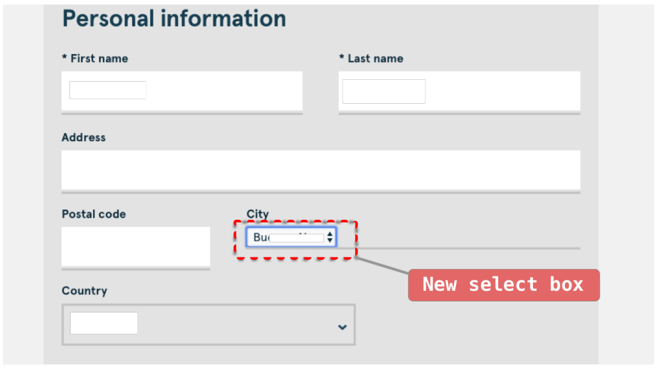


Fig. 1. The select box added by *Turn Input into Select* without adapting its style to the target page.

In order to minimize the alterations of the page's layout and aesthetics that CSWRs may cause, we have enriched the approach with a style adaptation mechanism to generate an appropriate style for the new elements added to the UI. The style adaptation is the final step of a CSWRs execution process and it basically consists in scanning certain elements in the page to find the style options for the new elements. Instead of focusing in finding the styles that suit best the target page, the mechanism looks for different alternatives in order to leave the final choice of style to designers, since it is important for them to be in control of the aesthetics. Section 4.2 shows an example of the style selection process in UX-Painter.

In order to generate the most appropriate styles for the widgets added by a CSWR, we have developed an algorithm that searches existing widgets of a certain type in the target page and extracts a subset of CSS properties from them. Both the type of widget and subset of CSS properties to choose from depend on the particular CSWR applied. For example, while the style detection algorithm for *Add Datepicker* looks for the color combinations most used in the page, the algorithm for *Add Link* captures the font and color of other links.

The algorithm does not depend on the execution of CSWRs, so it also could be used in other situations that require the scanning of certain widgets in a web page and retrieve their computed values of a given subset of CSS properties. Moreover, it can be configured with different search strategies which make it easily extensible. We have defined two strategies that are used in the style detection mechanism for each CSWR:

- *Distance Strategy*: finds all the widgets of a given type considering how close they are to the new element, and computes the values for their specified CSS properties. By sorting the existing widgets by euclidean distance to the new one, it is possible to capture the style of the similar widgets, prioritizing those that belong to the same page's section of the new one, and leaving the farther widgets as distant options. In this way, the most appropriate styles of the new widget correspond to those in the same section. For example, when a new link is added to the navigation bar of a page, it is more likely for the corresponding style to be the same as other links in the navigation bar, rather than the links in the page's footer. This strategy is used when the new widget has no predefined style, and it should look like the existing ones in the page. Such is the case for new links, text fields, select boxes and radio buttons. Figure 2 shows how the new select box of the previous example (see Figure 1) is adapted with the *Distance Strategy* .
- *Weight Strategy*: gets the values for the given CSS properties for the specified element types, sorting the results by weight of each value set, which is given by the number of occurrences within the entire page. Instead of capturing the style of the widget that surround a target element, this strategy allows to obtain the most used values for a set of CSS properties throughout the whole page. This way of capturing the target page's style is used when the new widget has a predefined style and only few aesthetic properties must be adjusted. For example, the CSWR *Turn Select into Autocomplete* (see section 4.2), through this strategy adjusts the colors and fonts used in the list of suggested values.

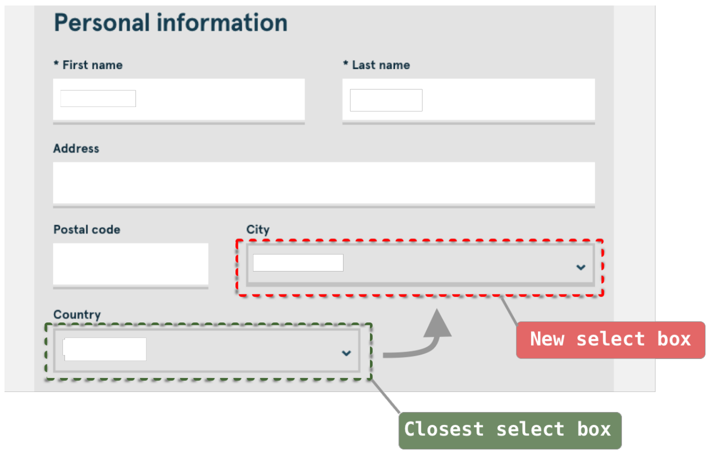


Fig. 2. The new select box added by *Turn Input into Select* captures the style of an existing select box through the *Distance Strategy*.

4 THE TOOL

4.1 Architecture

In order to satisfy the requirements mentioned in section 3.1, UX-Painter is based on the following components:

- The *Web browser* in which the rendered UI is modified without affecting the application's codebase. Given that the browser has been proposed as a mechanism to change the UIs externally, we use it to explore designs *in the wild* (in the context in which the application is used) and also without depending on the technology used to develop the target application.
- The *CSWRs palette*. Each CSWR described in section 3.2 can be applied and easily combined to achieve different design changes.
- *Application's versions* that are a composition of CSWRs. The versions provide the means to represent design changes that require the application of multiple CSWRs. Versioning the designs generated facilitates the remote testing, exploration and comparison of the design alternatives.

UX-Painter is implemented as a web-browser extension that works entirely on the client side, allowing the modification of a web application's UI through the application of CSWRs, which can be grouped and saved as an application version. Since the versions are persisted in the browser's local storage, the tool does not depend on any backend infrastructure. This architecture allows the usage of UX-Painter by just loading the extension on the browser, it does not require any configuration in the target application.

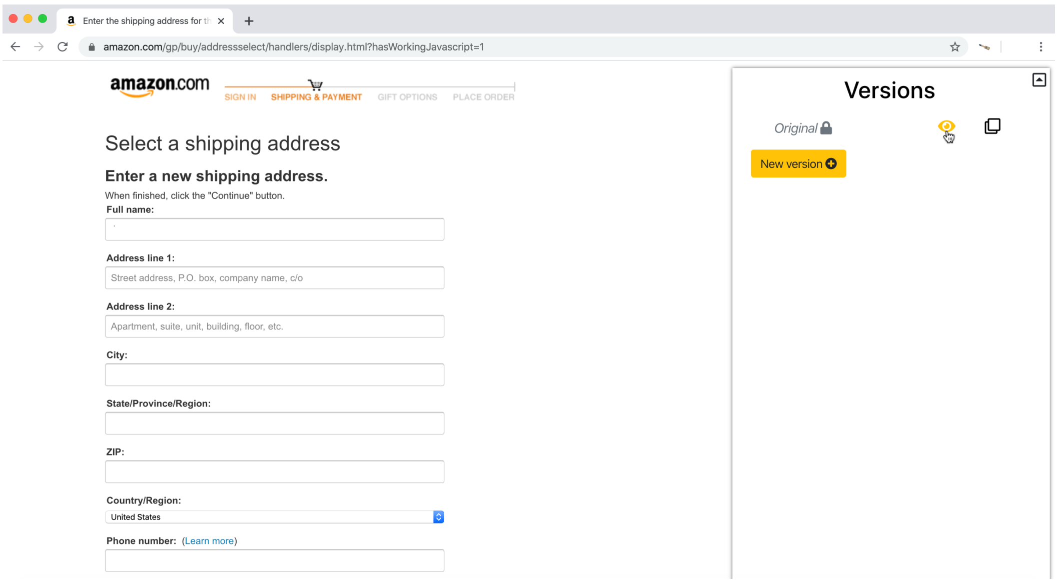
When creating a new version, UX-Painter persists, for each CSWR, all the necessary elements to recreate it: type, target element, user-defined parameters, the styles used, and the application's URL where it was applied. Along with the existing versions of the target web application, the tool also saves the currently active version. Whenever an application's page is loaded, the tool finds all the CSWRs of the active version and executes those matching the page's URL. If the user switches to another version, UX-Painter performs an undo of the CSWRs of the previous version and then executes the CSWRs of the new active version for the current page.

UX-Painter can be extended with new CSWRs which requires JavaScript and CSS programming skills to implement and integrate them to the tool. Each CSWR has two main functions: the transformation function that performs the underlying changes to the UI, and the one to discard these changes that is necessary to give to the tool's users the chance to cancel the CSWR. Both the transformation and the undo scripts must be implemented considering that a CSWR is meant to apply a very specific change to the UI, and that the functionality of the application does not have to be altered. Given the specificity of each CSWR, once it is designed, it should represent a short implementation time for developers.

4.2 UX-Painter in a nutshell

This section describes how the tool works with an example of usage in which some changes are performed in the Amazon's website.

The web-extension can be opened in any web page. Initially, as can be observed in Figure 3, it lists the existing application's versions indicating which one is currently being shown. By default there is only the original version, which cannot be edited to ensure that always a version is kept with the original design. Once a new version with a specific name is created, it can be edited with the application of different CSWRs on the page. Figure 4 illustrates the CSWRs palette that is shown when a new refactoring is added. Each available refactoring is listed along with an option that provides an extra description. As can be observed in Figure 5, the refactoring's information view is composed by a short description and two animations that make the refactoring's purpose clearer showing the UI before and after it is applied.


Enter the shipping address for th
amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1
amazon.com
SIGN IN
SHIPPING & PAYMENT
GIFT OPTIONS
PLACE ORDER
Versions
Original
New version

Select a shipping address

Enter a new shipping address.

When finished, click the "Continue" button.

Full name:

Address line 1: Street address, P.O. box, company name, c/o

Address line 2: Apartment, suite, unit, building, floor, etc.

City:

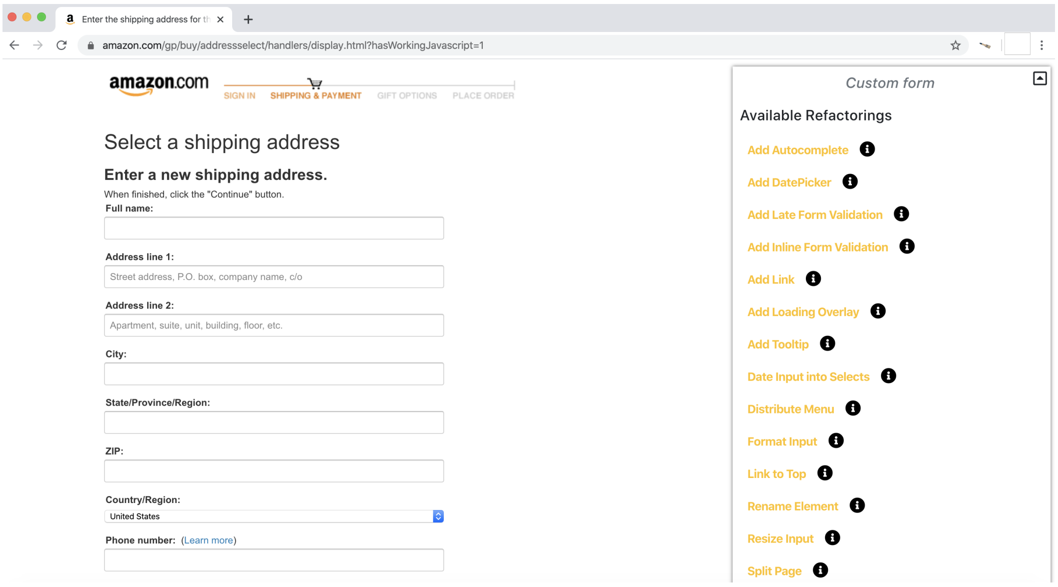
State/Province/Region:

ZIP:

Country/Region: United States

Phone number: (Learn more)

Fig. 3. Main view of UX-Painter. The original version is being shown.


Enter the shipping address for th
amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1
amazon.com
SIGN IN
SHIPPING & PAYMENT
GIFT OPTIONS
PLACE ORDER
Custom form

Select a shipping address

Enter a new shipping address.

When finished, click the "Continue" button.

Full name:

Address line 1: Street address, P.O. box, company name, c/o

Address line 2: Apartment, suite, unit, building, floor, etc.

City:

State/Province/Region:

ZIP:

Country/Region: United States

Phone number: (Learn more)

Available Refactorings

- Add Autocomplete
- Add DatePicker
- Add Late Form Validation
- Add Inline Form Validation
- Add Link
- Add Loading Overlay
- Add Tooltip
- Date Input into Selects
- Distribute Menu
- Format Input
- Link to Top
- Rename Element
- Resize Input
- Split Page

Fig. 4. Palette of CSWRs. The application process of a CSWR starts by selecting the desired one from the list.

In the Amazon's checkout form, a validation for mandatory fields can be added to minimize failed form submissions because of incomplete information. There are two refactorings that apply

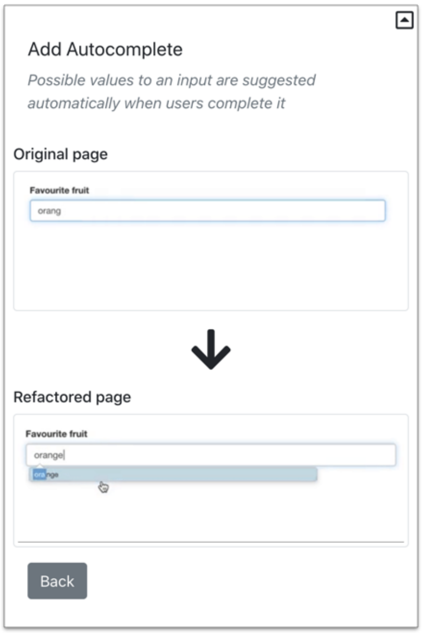
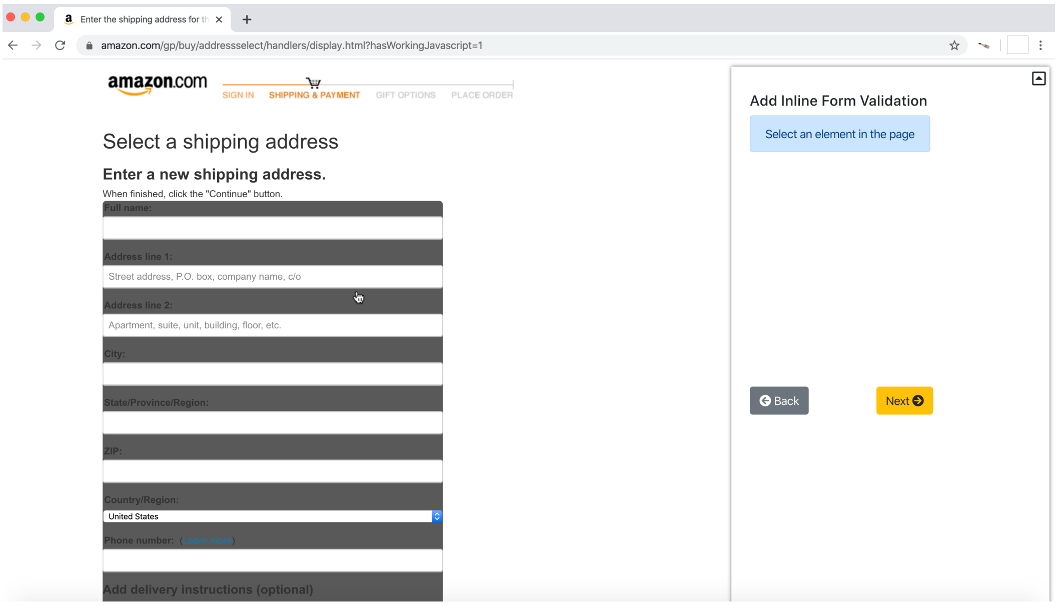


Fig. 5. Example of a CSWR's description page.

different types of validations: inline or late validation. In this example *Add Inline Form Validation* will be applied, but the application process of both refactorings is practically the same. After the selection of the desired refactoring from the palette, the tool guides the user through its application step by step. First, it enables the selection of the target element to be refactored directly over the page (Figure 6). Since each CSWR has its own type of target element, only elements that match the required type may be selected (in this case, the forms are the target element). Next, additional parameters needed to execute the CSWR may be requested. *Add Inline Form Validation* requires the mandatory form fields, so they must be selected in the page as shown in Figure 7. It is worth to mention that none of the available refactorings require interactions with any server, they work entirely on the client-side requesting the information needed to the users. The final step of the application process for all CSWRs shows a preview of the transformed page where the user can confirm or cancel the change (see Figure 8). In the case of *Add Inline Form Validation*, whenever the focus of a mandatory field is lost leaving it empty, a red border will appear indicating that a value is expected. After confirmation, the refactoring can be observed in the list of refactorings applied on the current version, marked with an asterisk to show that it has not been persisted and must be saved before the page is abandoned.

Following the process described before, the version may be incrementally edited with new refactorings to achieve larger design changes. For instance, in the checkout form already modified by *Add Inline Form Validation*, the country select box has too many options so a potential solution is to replace it with a text input with auto-complete through *Turn Select into Autocomplete*. This specific CSWR does not require user-defined parameters since the values that will be suggested are the select box options, so once the target element is selected, the preview of the refactoring is shown. In the preview, for those CSWRs like this one that add or replace widgets, UX-Painter allows choosing a style for them amongst a set of options. The tool executes the style adaptation algorithm described in Section 3.3 to show a short list with the N most appropriate styles (by default


Enter the shipping address for th
amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1
amazon.com
SIGN IN
SHIPPING & PAYMENT
GIFT OPTIONS
PLACE ORDER
Add Inline Form Validation
Select an element in the page
Back
Next

Select a shipping address

Enter a new shipping address.

When finished, click the "Continue" button.

Full name:

Address line 1: Street address, P.O. box, company name, c/o

Address line 2: Apartment, suite, unit, building, floor, etc.

City:

State/Province/Region:

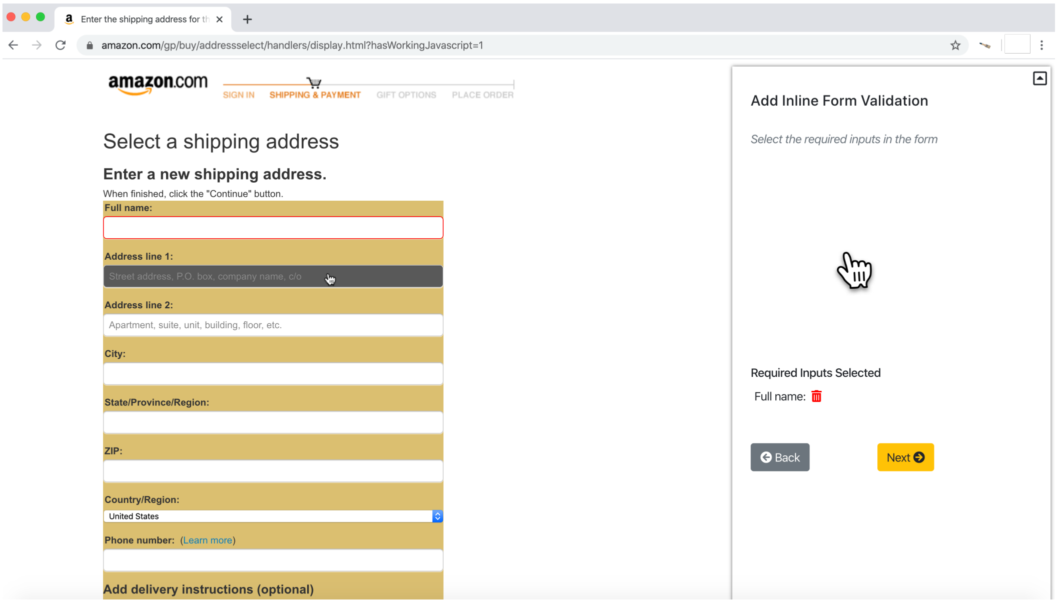
ZIP:

Country/Region: United States

Phone number: (Learn more)

Add delivery instructions (optional)

Fig. 6. The user must select the target form in the page. In this case, only forms are allowed.


Enter the shipping address for th
amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1
amazon.com
SIGN IN
SHIPPING & PAYMENT
GIFT OPTIONS
PLACE ORDER
Add Inline Form Validation
Back
Next

Select a shipping address

Enter a new shipping address.

When finished, click the "Continue" button.

Full name:

Address line 1: Street address, P.O. box, company name, c/o

Address line 2: Apartment, suite, unit, building, floor, etc.

City:

State/Province/Region:

ZIP:

Country/Region: United States

Phone number: (Learn more)

Add delivery instructions (optional)

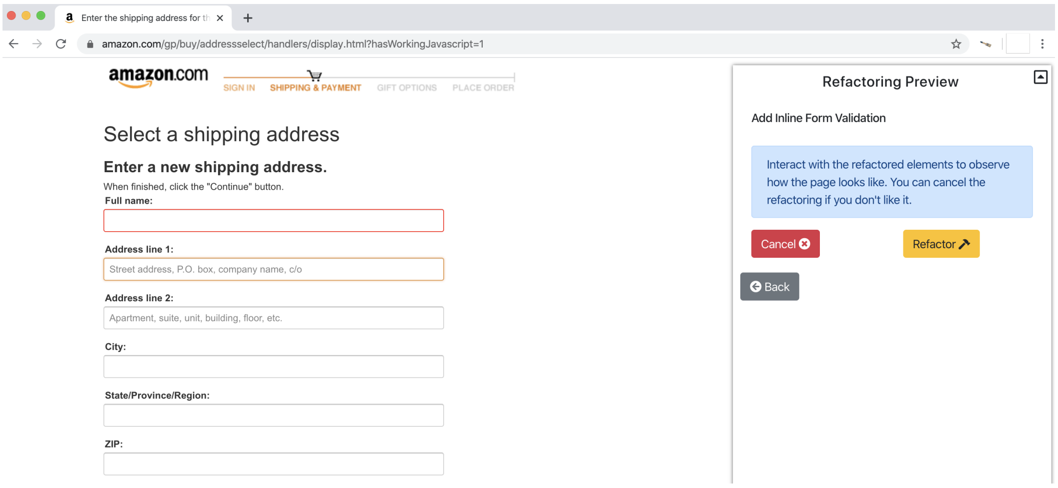
Select the required inputs in the form

Required Inputs Selected

Full name:

Fig. 7. UX-Painter waits for the selection of the mandatory form fields.

N=5, but it can be changed) for the new widget(s) and the user may instantaneously check how each one looks like and choose the preferred one. In the case of *Turn Select into Autocomplete* (see Figure 9), the user can inspect different styles to adjust the color of the list of suggested values.


Enter the shipping address for th

amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1

SIGN IN SHIPPING & PAYMENT GIFT OPTIONS PLACE ORDER

Select a shipping address

Enter a new shipping address.

When finished, click the "Continue" button.

Full name:

Address line 1: Street address, P.O. box, company name, c/o

Address line 2: Apartment, suite, unit, building, floor, etc.

City:

State/Province/Region:

ZIP:

Refactoring Preview

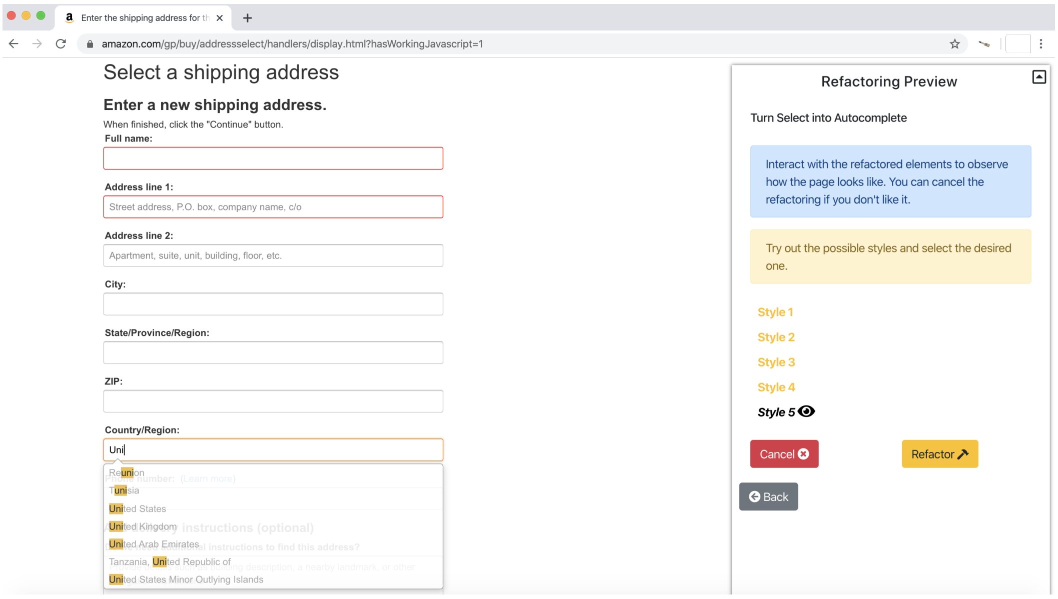
Add Inline Form Validation

Interact with the refactored elements to observe how the page looks like. You can cancel the refactoring if you don't like it.

Cancel · Refactor · Back

Fig. 8. The user can interact with the form to observe the result of the validation added.

Others refactorings that could help to improve the design of Amazon's checkout page are *Turn Attribute into Link* to create a link in the page's logo to go to the home page, *Resize Input* to modify the length of the ZIP field and *Rename Element* to change its label to "Postal Code". Figure 10 shows the new version as the result of the sequence of all CSWRs mentioned. The result can be compared with the default version observed in Figure 3. At any time, the user can start using a version by selecting it in the versions' list. Another version could be created to evaluate alternative CSWRs, for instance to test how the form works with late validation.


Enter the shipping address for th

amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1

Select a shipping address

Enter a new shipping address.

When finished, click the "Continue" button.

Full name:

Address line 1: Street address, P.O. box, company name, c/o

Address line 2: Apartment, suite, unit, building, floor, etc.

City:

State/Province/Region:

ZIP:

Country/Region: Uni

Reunion
Tunisia
United States
United Kingdom
United Arab Emirates
Tanzania, United Republic of
United States Minor Outlying Islands

Refactoring Preview

Turn Select into Autocomplete

Interact with the refactored elements to observe how the page looks like. You can cancel the refactoring if you don't like it.

Try out the possible styles and select the desired one.

Style 1
Style 2
Style 3
Style 4
Style 5

Cancel · Refactor · Back

Fig. 9. In the Turn Select into Autocomplete, the user can choose between different styles for the list of suggested values.

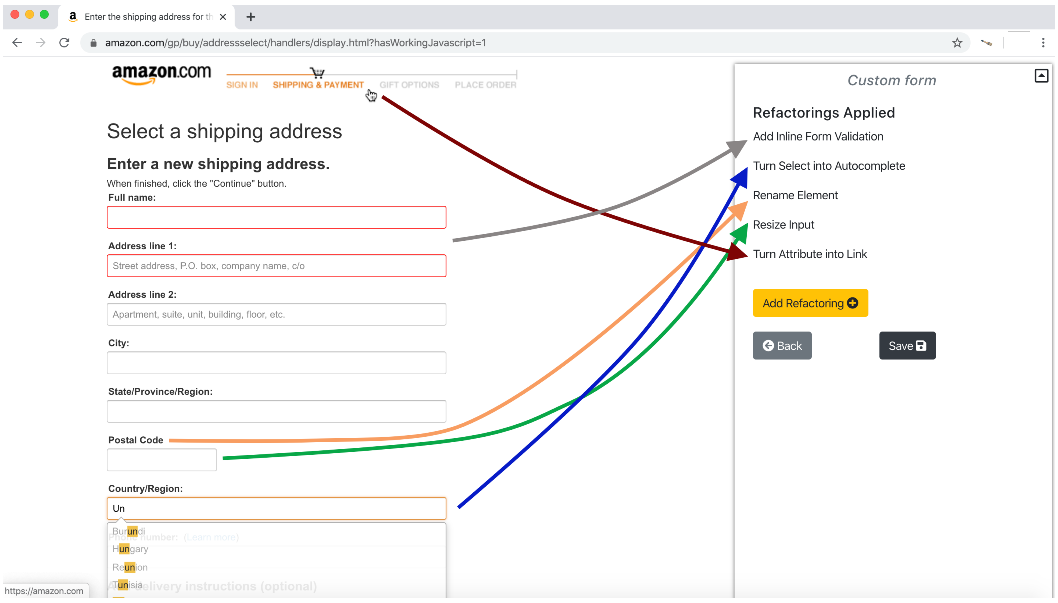

Enter the shipping address for th
amazon.com/gp/buy/addressselect/handlers/display.html?hasWorkingJavascript=1
amazon.com
SIGN IN
SHIPPING & PAYMENT
GIFT OPTIONS
PLACE ORDER
Select a shipping address
Enter a new shipping address.
When finished, click the "Continue" button.
Full name:
Address line 1:
Street address, P.O. box, company name, c/o
Address line 2:
Apartment, suite, unit, building, floor, etc.
City:
State/Province/Region:
Postal Code
Country/Region:
Un
Burundi
Hungary
Reunion
Tunisia
https://amazon.com
Custom form
Refactorings Applied
Add Inline Form Validation
Turn Select into Autocomplete
Rename Element
Resize Input
Turn Attribute into Link
Add Refactoring
Back
Save

Fig. 10. Alternative version of Amazon's checkout form. Each CSWR of this version it is linked with the performed change.

Note that a version groups all the application pages, which means that the user can go to another page and add new CSWRs to any of the existing versions. The version being used is also kept along the whole application, supporting navigation between the different pages. Other features included in UX-Painter are version cloning to facilitate the creation of alternative versions that share common changes, and the possibility to export a specific version that can be useful for a designer trying to run a remote user test to share a version with test subjects. Exporting a version generates a JSON file with all the necessary information to re-create the version in another browser. Next, the subjects can import the generated file in UX-Painter to load the shared version in their browser.

5 EXPERIMENTS

5.1 Preliminary evaluation

We performed a preliminary evaluation in a very first version of UX-Painter which did not include all the CSWRs of the palette (*Distribute Menu*, *Split Page* and *Turn Select into Autocomplete* were added later) and it did not provide a style adaptation for the changes performed. In this evaluation, we compared the CSWR application process using UX-Painter against manual coding with 12 students organized in 4 agile teams composed with 3 students each. The teams had to create alternative solutions for a set of problems found on the target application during user testing. For this purpose, 3 teams used UX-Painter and the remaining one acted as control group. Regarding the groups that used UX-Painter, the time required to apply the CSWRs ranged from 5 to 300 seconds (M=38, SD=63.25), while the control group that coded the CSWRs by hand took 17 minutes in average (MIN=5, MAX=30, SD=9.74).

Satisfaction questionnaires revealed that students felt positive about being able to apply changes so fast, but they found problems when they applied some of the CSWRs because their changes disrupted the page's design, deteriorating user interaction instead of making it easier. The results of this study gave us a valuable feedback to improve UX-Painter with features that we implemented later. Mainly, we detected the need to adapt the transformed widgets to the target application's style to effectively improve the UX. Moreover, we gained some insights to improve the tool's usability such as the incorporation of a clear description of each step of the CSWR application process and the preview of a CSWR to observe its result in the UI before confirming it. The students also proposed the addition of several new CSWRs that are planned for a future version of the tool.

5.2 Method validation

In order to gain feedback of the proposed approach we have conducted interviews with 16 professionals that work in the software development industry. From the total, 13 are UX/UI designers while the remaining 3 are UI developers. Within the first group, different roles were included, namely UI/Visual designers, interaction designers and UX researchers. The interviews had 3 parts: the first part consisted of a 3 questions to identify the practices and tools that subjects use to explore design changes in their projects; the second part consisted of a demonstration of UX-Painter with a real example, and in the third part subjects reflected on the applicability of the technique proposed by the tool in their current development practices through another 3 questions. We next report the results of each question asked together with the most relevant comments made by the subjects.

5.2.1 How often a developed UI is modified to improve its UX in the projects you are involved? The frequency with which a developed UI has to be changed in response to UX issues found (either in a testing or production environment) resulted very high. Some of the subjects commented that although they usually validate the UI with prototypes before its development, new issues arise during and after implementation, either because the client is not convinced with the result or users experience an unexpected behaviour. This allows to highlight the importance of conducting evaluations in the real application once it has been developed. Regarding the subject who answered "Never", she stressed that the lack of resources make it difficult for UX improvement after the implementation stage.

5.2.2 When a UX issue is discovered, do you evaluate alternative solutions? In this question, 11 subjects indicated that they usually evaluate alternative designs when a UX issue must be fixed. However, this does not mean that they always conduct formal evaluations to determine which solution is better. While the most used artifact to evaluate potential solutions are prototypes, sometimes they contemplate alternatives but they are not able to test them, and they end up implementing the one that seems the best guess.

5.2.3 How do you explore the potential solutions for the UX issues discovered? This question intends to determine if the subjects explore alternative designs directly in the working application. Only 5 subjects answered that they evaluate alternatives in the real UI using the browser inspection tools to perform the changes. However, one of them commented that the changes he is able to perform altering the rendered UI in the browser are limited and he needs the assistance of a developer to try out complex changes that require programming knowledge. Moreover, from that group of 5 subjects, 2 of them mentioned that they also could use some kind of prototype depending on the magnitude of the change to be evaluated. The most significant outcome of this question is that there is a clear majority of UX designers (11 out of 14) that only use prototypes to explore different designs, which evidences that UX designers mostly work in separate artifacts during the evaluation of design changes.

5.2.4 Beyond the technical aspects of UX-Painter, do you consider that the proposed method for exploring alternative designs can facilitate the evaluation of the UX?. The answers to this question provided valuable insights. A total of 13 subjects agreed that this approach could be very useful to explore some changes of a UI that must be adjusted right after its development or even when the UI has been released to solve issues experienced by the users. One of these subjects also explicitly mentioned that the method can be "realistic and faster than a prototype" referring to the possibility of evaluating changes in the real application. Regarding the way used to apply the changes through CSWRs, they highlighted the speed to apply changes and the availability of alternative CSWRs for a specific purpose. However, 2 subjects commented that the CSWRs are very specific to certain kind of changes. With respect to the 3 subjects who do not consider that this method can facilitate the evaluation of UX, they think that it would be difficult to integrate a new tool in the workflow that they have already defined, in which they use prototyping tools to validate the UIs.

5.2.5 What are the limitations of the proposed method? The main limitations found by the subjects are related with the CSWRs offered. They commented that there may be changes that they need to perform that are not in the CSWRs palette. For instance, style changes intended to modify the page's aesthetics (colors, fonts), or other changes that involve layout alteration to improve the user's navigation such as adding, moving and removing certain elements. Subjects also mentioned as a limitation that some CSWRs could not be well integrated in some web applications producing disruptions in the target UI. However, this is related with the implementation of the CSWRs rather than the exploration method. Apart from the CSWRs, as the subjects were asked to assess the feasibility of the method to evaluate the UX in general, 2 subjects reflected that UX goes beyond a user using an interface, and the method does not seem to provide support for those evaluations where a UI is not involved.

5.2.6 Would you make any other changes to UX-Painter to improve its overall usability? Regarding the possible changes to the implemented tool, considering that the palette of CSWRs can be expanded, 2 subjects suggested that it may be easier to perform a CSWR if the user first selects the element to be refactored and then chooses between the applicable CSWRs on it. Their argument is that the user will have a better understanding of the possible CSWRs that can be applied to each UI element. Moreover, the selection of the style options in the CSWR preview was a little confusing since it is not easy to foresee the differences between the options offered. Nice-to-have features mentioned by some subjects are the possibility to apply a CSWRs to several similar UI elements at once to avoid repetitive CSWR applications, and to allow exporting the code executed by the CSWRs when they have to be implemented.

5.3 Usability evaluation of UX-Painter

Besides the feasibility of the proposed technique, we ran a set of user tests to assess the overall usability of UX-Painter, this time with non-expert users. Given that UX-Painter does not require programming skills, a non-expert user instructed with the changes that need to be applied should be able to perform them on their own. We considered that the expertise on UX should not influence in the tool usage, but only gives designers certain criteria to define which changes should be explored. For this reason, the evaluation with non-expert users allow us to focus only in the usability of the tool without judging the changes that UX-Painter allows to perform.

Out of a total of 14 participants, 7 of them were lawyers, masters students (4 female, all between 30 and 40 years old), and the remaining 7 were computer-science researchers and postgraduate students (2 female, all aged between 25 and 60). During the validation, we gave each participant a website and we presented 3 UX problems along with one potential solution for each one. We decided to use well-known websites so the participants can easily understand the issues to solve and their possible solutions. In particular, the computer scientists used a Decision-Support System created by their own research group, which allows them to learn about UX improvements that they could implement to their system. On the other hand, the lawyers group used Amazon, one of the most popular e-commerce platform. After they understood the problems, the task they had to perform was to select and apply a CSWR for each of the problems using UX-Painter. The computer-science group had to divide a cluttered page into two sections (*Split Page*), replace a large select box with a text field (*Turn Select into Autocomplete*) and change a button's label (*Rename Element*). Meanwhile, the lawyers group had to change the label of a form's field (*Rename Element*), replace a large select box (*Turn Select into Autocomplete*) and convert a static image into a link (*Turn Attribute into Link*). After completing the task, we gave them a standard SUS questionnaire to gather satisfaction ratings. All sessions were approximately 10' long each.

The SUS questionnaire revealed a 85/100 satisfaction score (MIN=72.5, MAX=92.5, SD=5.3), which confirmed our impression that the subjects found UX-Painter useful and easy to use. Only with a brief demonstration of the tool in which an application of a CSWR was shown, they could understand how the tool works and they applied all the CSWRs for the UX problems provided. Subjects found the correct CSWR to apply very easily, except for *Turn Attribute into Link* and *Turn Select into Autocomplete* that took more time because their names were not familiar to the subjects. They confused *Turn Select into Autocomplete* with *Add Autocomplete*, while in the case of *Turn Attribute into Link*, the "attribute" term made them think that the CSWR was not applicable to images. These issues with the name and the description of the CSWRs are in line with the comments of some UX designers, who in the interviews commented that *refactoring* is not a common term in the UX world. With this evaluation, we gained feeback to improve the name of many CSWRs, specially those that can be applied in different element's types such as *AddTooltip*, *Rename Element* and *Turn Attribute into Link*.

Besides the selection of the correct CSWR for a determined solution, after the application of all the CSWRs, subjects were gladly surprised at the design changes achieved in a few minutes.

6 DISCUSSION

The interviews conducted with UX professionals and developers were very insightful to identify current gaps in the collaboration between UX designers and developers that the proposed approach may help filling. In fact, more than 80% of the interviewed subjects responded positively to the usefulness of the method and tool, recognizing the importance of evaluating the UX directly in the working application. Almost 70% of subjects confirmed that a developed UI has to be modified very frequently in response to discovered UX issues, but they use prototypes to explore and evaluate potential solutions mainly because it is the cheapest resource they have at hand.

These findings allowed us to assess the motivation of this work, which is based on the lack of methods to support the evaluation of design changes directly on a working application. In this context, the opinions of UX professionals indicate that the proposed exploration method can help to quickly assess alternative designs of a UI under analysis. Nevertheless, further experiments should be conducted in the context of a development process to assess if the method can be easily adopted and integrated with current UX practices.

Concerning the CSWRs presented, the majority of them provide changes for web forms, which make them suitable for form-based applications but could be limited for others like those that are based on complex navigation or contain cluttered pages. However, the purpose of this work was not to demonstrate an exhaustive list of solutions for interaction issues, but to show the feasibility of the approach to freely explore design changes directly in the target application. The feasibility of this approach encourages us to proceed conducting other studies to identify potential CSWRs for different web applications that may need to be incorporated.

The user tests conducted on UX-Painter provided some insights to facilitate the creation of different application's versions. Beyond the mentioned aspects of UX-Painter that should be improved, one the main limitation of the current implementation of the tool is that it relies on the browser's page load event to re-create a specific version, trying to execute the complete CSWRs sequence that match the target URL. This mechanism for applying the changes of a version, may not work properly in modern Single Page Applications (SPA) where the pages' content changes dynamically without reloading the entire page. For this type of architecture, a strategy that keep track of all state changes of the target application must be developed in order to precisely determine when to recreate each of the CSWRs.

The fact that the tool was used by non-UX experts evidenced another potential applicability of UX-Painter, which is the creation of personalized views by end-users of third-party applications. This way, any user with no background in programming or development processes, could mitigate some of the UX problems they suffer by creating their own, personalized view of the application in the browser, playing with alternative CSWRs until they find the most suited for them. Since the new application's versions are saved in the browser's local storage, the CSWRs of the selected version will be reapplied every time the user navigates to the application with that browser. However, the utility of the tool for personalizing a UI was out of the scope of this work.

7 CONCLUSIONS AND FUTURE WORK

In this work we presented a method for exploring design alternatives directly on a web application's UI without the need of coding them. We described the need of evaluating UI changes in the working application to detect issues not found in previous design stages. We have also conducted interviews with professional designers, assessing the lack of tool support to synchronize their practices with developers when a UI has been developed and UX improvement should take place. The CSWR technique serves designers to try out design changes, since it does not depends on the underlying technology of the target application, which makes it capable of modifying any web application's UI. In spite of the potential use of the technique, designers also revealed that the presented CSWRs address a restricted set of interaction fixes, so future work includes the addition of new CSWRs.

Regarding the impact of the changes on the page aesthetics, we incorporated to the method a style adaptation process to align the changes performed by the CSWRs to the target page's look and feel, and to reduce the issues that may arise in the layout. Although the algorithm developed showed promising results, further evaluations are required to assess its overall validity.

We implemented the proposed approach in UX-Painter, a web-extension that allows to apply the CSWRs in a WYSIWYG fashion. The new designs can be persisted and shared among different web browsers, which makes the tool suitable for remote user testing. Experiments with non-UX experts demonstrated the usability of the tool.

Beyond the incorporation of new CSWRs, we are also planning more experiments in the context of agile development teams, and also exploring the value that UX-Painter may offer to developers, by assisting them in the implementation of the transformations in the backend.

ACKNOWLEDGMENTS

Authors of this publication acknowledge the support from the Argentinian National Agency for Scientific and Technical Promotion (ANPCyT), grant number PICT-2015-3000, and the contribution of the Project 691249, RUC-APS: Enhancing and implementing Knowledge based ICT solutions within high Risk and Uncertain Conditions for Agriculture Production Systems (www.ruc-aps.eu), funded by the European Union under their funding scheme H2020-MSCA-RISE-2015.

REFERENCES

[1] Iñigo Aldalur, Marco Winckler, Oscar Díaz, and Philippe Palanque. 2017. Web Augmentation as a Promising Technology for End User Development. In *New Perspectives in End-User Development.* Springer International Publishing, Cham, 433–459. https://doi.org/10.1007/978-3-319-60291-2{_}17

[2] Kent Beck and Cynthia Andres. 2005. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, Boston, Massachusetts, USA.

[3] Manuel Brhel, Hendrik Meth, Alexander Maedche, and Karl Werder. 2015. Exploring principles of user-centered agile software development: A literature review. *Information and Software Technology* 61 (2015), 163–181. https://doi.org/10.1016/j.infsof.2015.01.004

[4] Tonio Carta, Fabio Paternò, and Vf De Santana. 2011. Web usability probe: a tool for supporting remote usability evaluation of web sites. In *Human-Computer Interaction – INTERACT 2011. LNCS 6949*. Springer Berlin Heidelberg, Berlin, Heidelberg, 349–357. https://doi.org/10.1007/978-3-642-23768-3{_}29

[5] Lily Cho. 2009. Adopting an agile culture: A user experience team's journey. In *Proceedings - 2009 Agile Conference, AGILE 2009*. IEEE, Chicago, IL, USA, 416–421. https://doi.org/10.1109/AGILE.2009.76

[6] Tiago Silva Da Silva, Angela Martin, Frank Maurer, and Milene Silveira. 2011. User-centered design and agile methods: A systematic review. In *Proceedings - 2011 Agile Conference, Agile 2011*. IEEE, Salt Lake City, UT, USA, 77–86. https://doi.org/10.1109/AGILE.2011.24

[7] Tiago Silva Da Silva, Milene Selbach Silveira, Frank Maurer, and Fábio Fagundes Silveira. 2018. The evolution of agile UXD. *Information and Software Technology* 102, July (2018), 1–5. https://doi.org/10.1016/j.infsof.2018.04.008

[8] Oscar Díaz, Cristóbal Arellano, Iñigo Aldalur, Haritz Medina, and Sergio Firmenich. 2016. Web Mashups with WebMakeup. In *Rapid Mashup Development Tools*. Springer International Publishing, 82–97. https://doi.org/10.1007/978-3-319-28727-0_6

[9] O. Díaz, C. Arellano, and M. Azanza. 2013. A language for end-user web augmentation: Caring for producers and consumers alike. *ACM Transaction on the Web* 7, 2 (2013), 1–51. https://doi.org/10.1145/2460383.2460388

[10] Damiano Distante, Alejandra Garrido, Julia Camelier-Carvajal, Roxana Giandini, and Gustavo Rossi. 2014. Business processes refactoring to improve usability in E-commerce applications. *Electronic Commerce Research* 14, 4 (2014), 497–529. https://doi.org/10.1007/s10660-014-9149-0

[11] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Professional, Boston, Massachusetts, USA. 431 pages.

[12] Krzysztof Z. Gajos, Daniel S. Weld, and Jacob O. Wobbrock. 2010. Automatically generating personalized user interfaces with Supple. *Artificial Intelligence* 174, 12-13 (2010), 910–950. https://doi.org/10.1016/j.artint.2010.05.005

[13] Andrei Garcia, Tiago Silva da Silva, and Milene Selbach Silveira. 2019. Artifact-facilitated communication in agile user-centered design. In *Lecture Notes in Business Information Processing*, Vol. 355. Springer Verlag, 102–118. https://doi.org/10.1007/978-3-030-19034-7{_}7

[14] Alejandra Garrido, Sergio Firmenich, Gustavo Rossi, Julian Grigera, Nuria Medina-Medina, and Ivana Harari. 2013. Personalized Web Accessibility using Client-Side Refactoring. *IEEE Internet Computing* 17, 4 (7 2013), 58–66. https://doi.org/10.1109/MIC.2012.143

[15] G. Ghiani, F. Paternò, L.D. Spano, and G. Pintori. 2016. An environment for end-user development of web mashups. *Int. Journal of Human-Computer Studies* 87 (2016), 38–64. https://doi.org/10.1016/j.ijhcs.2015.10.008

[16] Julián Grigera, Alejandra Garrido, Jose Ignacio Panach, Damiano Distante, and Gustavo Rossi. 2016. Assessing refactorings for usability in e-commerce applications. *Empirical Software Engineering* 21, 3 (2016), 1224–1271. https://doi.org/10.1007/s10664-015-9384-6

[17] Julián Grigera, Alejandra Garrido, José Matías Rivero, and Gustavo Rossi. 2017. Automatic detection of usability smells in web applications. *International Journal of Human-Computer Studies* 97 (2017), 129–148. https://doi.org/10.1016/j.ijhcs.2016.09.009

[18] Julián Grigera, Alejandra Garrido, and Gustavo Rossi. 2017. Kobold: Web Usability as a Service. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 990–995. https://doi.org/10.1109/ASE.2017.8115717

[19] Applause App Quality Inc. 2019. uTest - The Professional Network for Testers. https://www.utest.com/

[20] Gabriela Jurca, Theodore D. Hellmann, and Frank Maurer. 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*. IEEE, 24–32. https://doi.org/10.1109/AGILE.2014.17

[21] Andrew J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The state of the art in end-user software engineering. *Comput. Surveys* 43, 3 (2011), 1–44. https://doi.org/10.1145/1922649.1922658

[22] Ron Kohavi and Roger Longbotham. 2016. Online Controlled Experiments and A/B Testing. *Encyclopedia of Machine Learning and Data Mining* 7, 8 (2016), 1–8. https://doi.org/10.1007/978-1-4899-7502-7

[23] Kati Kuusinen. 2016. BoB: A Framework for Organizing Within-Iteration UX Work in Agile Development. In *Integrating User-Centred Design in Agile Development*. Springer International Publishing, 205–224. https://doi.org/10.1007/978-3-319-32165-3{_}9

[24] M. Nebeling, M. Speicher, and M. C. Norrie. 2013. W3touch: Metrics-Based Web Page Adaptation for Touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 2311–2320. https://doi.org/10.1145/2470654.2481319

[25] Vagner Figueredo de Santana and Maria Cecília Calani Baranauskas. 2015. WELFIT: A remote evaluation tool for identifying Web usage patterns through client-side logging. *International Journal of Human-Computer Studies* 76 (2015), 40–49. https://doi.org/10.1016/j.ijhcs.2014.12.005

[26] Desirée Sy. 2007. Adapting usability investigations for agile user-centered design. *Journal of Usability Studies* 2, 3 (2007), 112–132.

[27] Test IO. 2019. QA Testing as a Service | test IO. https://test.io/

[28] J Wong and J Hong. 2007. Making Mashups with Marmite: Towards End-User Programming for the Web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. Association for Computing Machinery, New York, NY, USA, 1435–1444. https://doi.org/10.1145/1240624.1240842

Received October 2019; revised November 2019; revised March 2020; accepted April 2020