

Umitation: Retargeting UI Behavior Examples for Website Design

Yan Chen

yanchen@dgp.toronto.edu
University of Toronto
Toronto, Ontario, Canada

Tovi Grossman

tovi@dgp.toronto.edu
University of Toronto
Toronto, Ontario, Canada

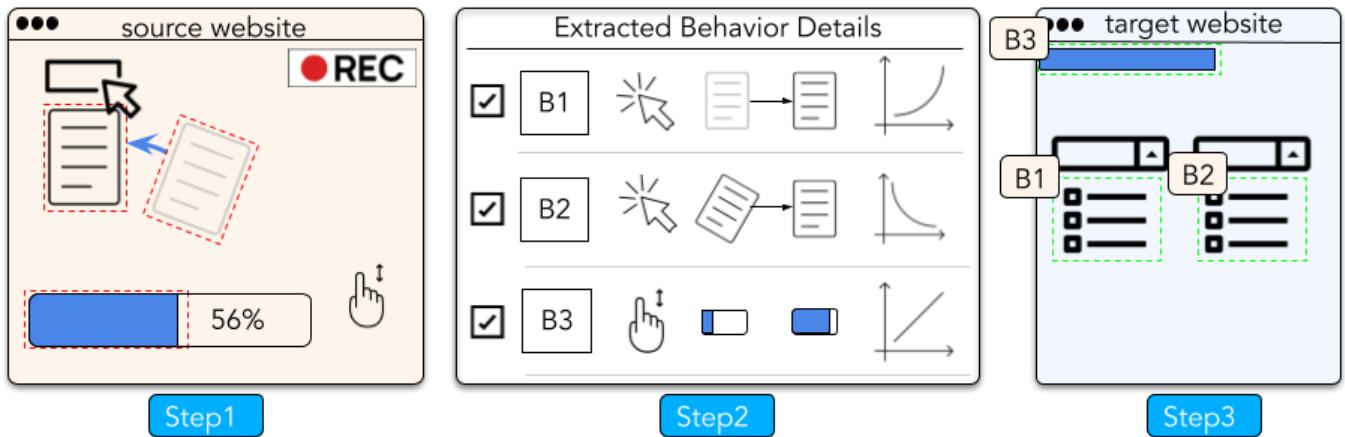


Figure 1: The workflow of Umitation. There are three steps to using Umitation to retarget example UI behaviors from a source website to a target website. (Step 1) Users first specify one or more source elements on the source website and then record their behaviors by interacting with them. Umitation will automatically capture the Document Object Model (DOM) changes. (Step 2) Umitation displays the low-level details of the recorded behaviors on its main panel, and users can directly manipulate the meta data of the behaviors. (Step 3) Umitation guides users to retarget the behaviors to appropriate elements (e.g., structurally similar elements) on the target website.

ABSTRACT

Interface designers often refer to UI behavior examples found in the wild (e.g., commercial websites) for reference or design inspiration. While past research has looked at retargeting interface and webpage design, limited work has explored the challenges in retargeting interactive visual behaviors. We introduce Umitation, a system that helps designers extract, edit, and adapt example front-end UI behaviors to target websites. Umitation can also help designers specify the desired behaviors and reconcile their intended interaction details with their existing UI. In a qualitative evaluation, we found evidence that Umitation helps participants extract and retarget dynamic front-end UI behavior examples quickly and expressively.

KEYWORDS

Retarget design, UI behavior examples, user intent and disambiguation

ACM Reference Format:

Yan Chen and Tovi Grossman. 2021. Umitation: Retargeting UI Behavior Examples for Website Design. In *The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21)*, October 10–14, 2021, Virtual Event, USA. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3472749.3474796>

1 INTRODUCTION

Interactive behavior, defined by Myers et al. [48] as the “feel” (as opposed to the “look”) of a user interface (UI), is a key component of the modern website. When designed well, interactive UI behaviors can make a website engaging and user-friendly. UI designers often browse behavior examples and use them on their own artifacts to explore different aspects of the design [22, 50]. Exploring existing solutions that fit the current context can help designers creatively address unfamiliar situations [30, 31]. In-context interactive mock-ups could also help make team communication more effective [11]. However, current practices for leveraging design examples on a new interface have four main limitations: they are often 1) informal, with designers using professional tools (e.g., Adobe XD, Figma) with a limited set of pre-defined modules; or 2) ad hoc, with designers

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

UIST '21, October 10–14, 2021, Virtual Event, USA

© 2021 Association for Computing Machinery.

ACM ISBN 978-1-4503-8635-7/21/10...\$15.00

<https://doi.org/10.1145/3472749.3474796>

saving multiple versions of files with cryptic file names to indicate their relevance [50]; or 3) time-consuming, with designers viewing and modifying the source code from the example websites [36]; or 4) ambiguous, with designers omitting critical design contexts when collaborating with developers [38, 48].

Past research has looked at retargeting UI visual design [33, 56], but the process of retargeting interactive and dynamic behaviors remains unaddressed due to their high complexity and dynamic nature. When retargeting example UI behaviors to their own interfaces, designers face several challenges. First, modern UI behaviors (e.g., responsive effects) often involve multiple response elements, each of which may have dynamic effects present across several CSS attributes. This makes the behavior difficult to track, understand, and adapt to a separate UI. Second, in the later stages of UI development, designers' interfaces can be arbitrarily complex, with many pre-built elements and behaviors. This makes it tedious to specify the low-level details when experimenting with example behaviors (e.g., identifying the trigger object and response objects) and to keep track of behavior variants. Third, the implementation of the example UI behaviors can require domain expertise in specific frameworks and libraries, which designers may not have. These limitations led to our research question: ***How can we make it easier for designers to retarget UI behavior examples to their own interfaces?***

In this paper we introduce Umitation,¹ an interactive system that allows interface designers to **easily extract** and **efficiently retarget** dynamic UI behavior examples from existing (source) websites to their own (target) websites. These examples can be new effects the users want to experiment with, or they can act as learning materials that help users understand the functional relationships between objects in the example behaviors. We designed Umitation to retarget only the front-end visual behaviors that occur within the DOM tree of a website, rather than the behaviors' underlying computational functions. This design can help designers avoid the complexity of detailed implementation by providing a low barrier to entry and contextualized guidance to help specify details while maintaining considerable expressiveness. It is also designed in accordance with what David Kelley terms “enlightened trial and error,” in which prototypes are fluidly produced and evaluated.

Led by prior research, we designed Umitation with three goals to help users easily and efficiently retarget UI behavior examples they find in the wild (more details in Section 2).

- **DG1: Quick to extract UI behavior:** Designers need to quickly extract and manage the dynamic UI behavior examples found in the wild for further usage [8, 19, 36].
- **DG2: Easy to understand and modify the example behaviors:** Designers need to understand the interaction components in the recorded examples and be able to manipulate and remix them for further use [11, 23].
- **DG3: Easy to experiment with multiple example behaviors on different interfaces:** Designers should be able to easily retarget variants and alternatives of the example behaviors in other UI contexts [20, 48, 50].

¹Umitation is an acronym for User interface behavior imitation.

To achieve these goals, Umitation elicits and decomposes an interface behavior into four distinct components: *trigger*, *trigger object*, *response*, and *response object*. Consider the example illustrated in Figure 1, in which a user wishes to retarget the behavior of a dropdown menu. In this example, these four components correspond to a mouse click (*trigger*), the menu button (*trigger object*), the dropdown list's fade-in (*response*), and the dropdown list itself (*response object*). Figure 1 shows three steps of using Umitation. (Step 1) To specify which example behaviors to extract, users first select one or more source elements on the source website, and then they record the elements' behaviors by interacting with them. Umitation will automatically capture the Document Object Model (DOM) changes (e.g., via MutationObserver). (Step 2) To help users repurpose the example behaviors, Umitation unpacks the behavior details (e.g., relationships between the trigger objects and response objects) and tabulates them on its main panel. This panel is also designed to reduce users' efforts in modifying the extracted behaviors by allowing them to directly manipulate and remix the behaviors' meta data. (Step 3) To help specify the target elements on the user's website, Umitation automatically finds elements on the target websites that are structurally similar (e.g., position, size) to those on the source websites. By adding switch buttons (i.e., toggle the event listeners added on the elements) associated with each behavior of the target elements (e.g., B1, B2, B3 of Step 3), Umitation reduces the context-switching effort of experimenting with multiple versions of behaviors.

We conducted a user study with eight experienced UI designers to evaluate Umitation's usability and efficiency. We found that participants could use Umitation to successfully extract and retarget five different types of UI behavior examples. The participants found that Umitation helped them understand the behaviors more easily by unpacking the black box of complex UI behaviors, disambiguated the original abstract intent by guiding them to specify the low-level details, and saved them the effort of searching the source code and identifying implementations. Additionally, although our tool was not formally compared to baseline approaches, participants reported that Umitation could help them save time in an order of magnitude compared to their current practices. In the final section of this work, we discuss future work that can adapt our approach to other types of interactive behaviors.

This work is an important step towards the vision of real-time, example-based dynamic behavior retargeting with a focus on user intent disambiguation. We make the following contributions:

- A set of novel interaction designs and techniques that allow users to easily extract, manipulate, and retarget dynamic UI behavior examples and their variants to other websites.
- Umitation, an end-to-end system that integrates these techniques, along with a user study showing its usability and effectiveness in helping make interface exploration easy and efficient.

2 RELATED WORK

Umitation builds on decades of rapid UI prototyping tools and demonstrates how users can easily retarget complex and dynamic interactive behaviors. It also extends the literature on user intent disambiguation interfaces with functional, intelligent behaviors. In

this section, we discuss prior work in example retargeting, functional copy and paste, and record and reuse.

2.1 Example-Based Retargeting for Interface Design

Rapid UI prototyping is a well-recognized and common practice throughout the design lifecycle [4]. It helps people to more quickly get early feedback on designs and capture their fleeting ideas. Early research efforts in supporting rapid prototyping were mainly spent on reducing the overhead costs of sketching [34, 43]. As the Web now offers a corpus of example resources, more recent efforts have focused on helping designers make use of these examples in their work. Using examples of previous work and knowledge is an established technique [6] that can provide designers potential options for modeling and inspiration. Viewing existing designs is also a helpful step in sparking creativity [3].

Prior work has explored methods of helping people more efficiently search examples [36], suggest alternatives, compare multiple versions [45], and even auto-generate examples. For example, Lee et al. [36] showed that designers prefer an adaptive example selection process using interface metadata for viewing and navigating example galleries. They also found that users made use of multiple example UIs when designing new websites. Dow et al. showed that sharing multiple designs could improve outcomes, encourage exploration, and enhance group rapport [16]. Multiple systems have explored the space of alternative design, especially within a mixed-initiative paradigm in which users and intelligent agents take turns refining the design details [47, 51, 55, 61]. Kumar et al. developed a series of models to tackle the problem of automatically retargeting web design examples. However, these techniques focused only on the interface's static appearance (e.g., page layout) without considering user behaviors, which are also part of the interaction and which use meta data that can be dynamic and non-trivial to specify and capture. Therefore, our first design goal (DG1) is to *quickly extract UI behavior examples*.

2.2 Enhanced Copy and Paste

After finding relevant examples, retargeting them for one's own purpose — which Gick and Holyoak term “analogy pervades thought” — is more efficient than reinventing them from scratch [17]. Our work was inspired by many prior techniques that allow users to copy and paste example objects from one place to another, which is the most common form of retargeting examples. Among them, the most commonly used is the text copy-and-paste technique, which allows users to create a duplicate of a word or a passage and place it somewhere else. More recently, as objects have become more complex, many enhanced copy-and-paste techniques have been proposed. Microsoft’s Format Painter supports copy-and-paste for object properties (e.g., colors, fonts) [1], Citrine [54] enables users to structurally copy and paste text data (e.g., mailing addresses) across different applications, and much work in the machine-learning community has focused on exploring how to transfer the properties or skeletons of content, such as body poses in videos [7].

2.3 Connecting UI Design and Implementation

In the context of UI design, several systems have proposed techniques to help retrieve a design’s associated code [24, 46], teach the relevant implementation [42], create web mashups [29, 60], or implement code using existing app functions [29]. WebCrystal [8] helps users to inspect and learn from existing UI layout implementations and later recreate their own, but it focuses only on the static layout and style aspects of UI design. Hirschman et al. developed multiple tools to ease the learning process for understanding the implementation of interactive UI behaviors [23, 24]. While these systems help *ease the process of understanding interactive behaviors* (DG2), which also guides our system design, they either focus on helping users learn the implementation or require them to know the implementation of the examples before repurposing.

2.4 Exploration and Prototyping

Our work differs from this prior work in a fundamental way: instead of easing the implementation learning and reuse process, Umitation is designed to help designers rapidly explore and prototype example UI behaviors found in the wild on their own interfaces. To achieve this goal, Umitation helps designers avoid detailed implementations by providing a low barrier to entry for UI behavior extraction, manipulation, and experimentation (Fig. 1). This design is partially inspired by several prior systems that allow users to *create UIs via direct manipulation* (DG2) and spreadsheet-like UIs. For instance, d.mix [19] allows users to create UIs by directly selecting example elements from existing websites, and the created UIs still connect with the websites (e.g., data gets updated). Gneiss [9] and Vegemite [44] let users create websites using a spreadsheet where the data links to pre-made functions. Prior work has shown that these approaches are effective at helping users to more easily compare multiple versions of designs when creating web UIs. *We followed this design guideline and have aimed to make it easy to use Umitation to experiment with multiple versions of the behaviors* (DG3).

2.5 Recording and Reuse

An important technical component of Umitation’s system is the UI behavior extraction technique (Fig. 1 from Step 1 to Step 2), which relates to the body of work using the record and reuse (R&R) technique. Unlike video recording, R&R captures and presents the rich meta data pertaining to the user’s operations (e.g., attribute changes, user input events), making data management and debugging more simple. Systems such as Scry [5], Doppio [14], and FireCrystal [52] use this technique to support designers in understanding UI behaviors. Other than web UI design, tools like Chronicle [18] and Aquamarine [49] capture and visualize the meta data of user operations on software applications, making their operation history easier to retrieve or manipulate. ReverseORC [25] supports retargeting the dynamic resizing behavior of a GUI across platforms. Systems like Montage [37] and Pronto [39] support VR/AR recording augmentation for new digital content prototyping. Text-based R&R technique is also commonly used for learning [2] and content updating [53]. Adding to this body of work, Umitation focuses on helping disambiguate user intent by using the R&R technique for

complex and dynamic UI behavior retargeting for main component specification and for the functional relationship between objects.

The workflow of record and reuse is also similar to that of the programming by demonstration (PbD) paradigm [15], where machines learn the task model from users' task performances (e.g., pick-and-place an object, navigate on a smart phone). Many prior systems, such as Rousillon [10] and Sugilite [40], have built on this paradigm to lower users' efforts in domain-specific task programming. Umitation also draws lessons from this paradigm and explores the design space of UI behavior example retargeting, with focuses on ease of use and user intent disambiguation.

3 UMITATION

With the three design goals introduced above, we created Umitation to help designers experiment with UI behavior examples found on other websites (the source website) on their own website (the target website) during web UI prototyping. Instead of inspecting the implementation or reinventing the code from scratch, they use Umitation to record how an element behaves and extract it into a self-contained *behavior object*. Once this is complete, they can edit the extracted behaviors to contextually experiment with them on their target website. In this section, we first illustrate the experience of using Umitation with a sample usage scenario that embodies many of the common UI behaviors found in prior work. We then detail the design and implementation of Umitation.

3.1 The Umitation User Experience

Sophie, a professional web interface designer, is in the middle of designing a website that already has some content and behaviors. Now she wants to make the site more engaging by adding two dynamic UI behaviors: 1) dropdown effect—when hovering over the dropdown menu, the dropdown list will appear with an animated effect, and 2) scroll progress—the progress bar on the top of the website will indicate the current scrolling percentage². She decides to use Umitation to leverage some example behaviors that she has already found. Sophie first clicks the Chrome extension on her browser to start Umitation. Similar to a text copy-paste function, Umitation prompts her to first select the elements that she is interested in copying on the source website: the dropdown menu list element and the progress bar. She then clicks the “Record” button (Fig. 2.A) to record a demonstration of her pointer hovering over the menu button to trigger the dropdown list animation and the act of scrolling down the website from top to bottom. Once finished, she sees three rows added to the table on Umitation’s main panel (Fig. 2).

To understand these example behaviors, Sophie first reads their descriptions (Fig. 2.E) and quickly scans through the basic elements of each behavior, including the trigger and trigger objects (Fig. 2.F) and the screenshots of the response object (Fig. 2.G) to understand the overall structure of the behavior. Then she analyzes the attribute relationship charts between the trigger object and response object (Fig. 4.H4) to see if they match the way she thinks the object should behave. By hovering over the points on each chart, she reads the values of the associated attributes. By clicking on the pre-built

²Similar to the top progress bar behavior on webflow.com/blog/parallax-scrolling while scrolling

functions, she modifies the extracted behaviors that she wants to test on her own website.

For the dropdown list behavior, Sophie is not sure if the extracted behavior will fit well on her website, so she decides to experiment with different variants to see which one is the most suitable. She clicks the “Variant” button (Fig. 2.H,) and sees a prompt (Fig. 2.J) that requires her to specify the number of variants and asks whether she wants to use pre-defined interpolation functions using the starting and ending points of the extracted value. She types the number three to create three variants and uses pre-defined interpolation functions, and Umitation adds three items on the “Others” column (Fig. 2.I) for the transformed attributes.

For the progress bar behavior, Sophie sees that the unit of the width is in pixels on the attribute relationship chart (Fig. 4.H4), but she finds it easier to think of the change in terms of a percentage. Therefore, she clicks the dropdown menu (Fig. 4.H3) and selects % as the unit. Immediately, Umitation prompts her to go back to the source website to select the element for which the width will be used as the percentage total. Sophie goes back to the source website and selects the progress bar’s background DIV, and the chart updates its y-axis by dividing the original width value by the width of the selected DIV.

To retarget the progress bar behavior to her website, Sophie first clicks the checkbox on its behavior row (i.e., Behavior 2) and clicks “Manually Select” (Fig. 2.C). Following Umitation’s step-by-step instructions, Sophie goes to the target website and selects the corresponding progress bar and background elements. By scrolling down the page, she sees how the progress bar synchronizes with the percentage of page scrolling. By reselecting the background element, she sees how the progress bar’s maximum value changes accordingly.

Similarly, for the dropdown menu behaviors, Sophie clicks the checkbox on the two dropdown menu animation behavior rows (i.e., Behavior 1, 2) and clicks “Auto Match” (Fig. 2.B) to save herself the effort of matching multiple behaviors. Once she has switched to her own website, Sophie is asked whether she wants to apply the variants to similar elements that were detected by the system. After confirming that option, she sees that the four different dropdown menu behaviors (the original one and the three variants) are added to the four dropdown menu buttons (Fig. 6.B). She hovers over each menu and views the different effects. After comparing these variants, Sophie decides she likes variant number three the best, so she records a short video to demonstrate the high-fidelity UI behavior and sends it to her teammates for feedback and implementation. All her extracted example behaviors were stored in Umitation and will continue to work even if their source websites change.

3.2 Design and Implementation

We implemented Umitation as a Chrome browser extension with 2,000 lines of JavaScript code. Primarily, we used jQuery to ease element selection and widget creation, Web APIs to watch for element changes and user input events, and a few other third-party libraries and scripts which we describe below. The next sections describe the major steps of the system workflow.

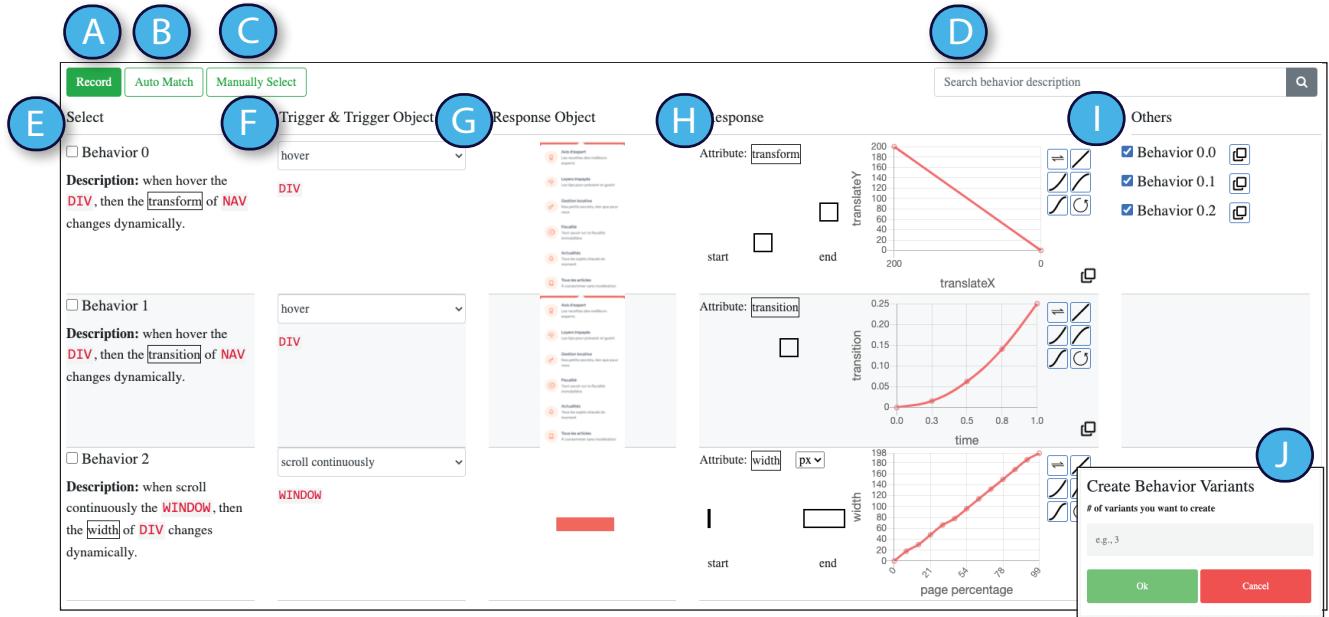


Figure 2: Umitation’s main panel (Step 2 in Fig. 1). The major view on this panel is a table that serves as a behavior clipboard, with each row showing one attribute change and each column (E–I) showing one interaction detail. A user can edit each interaction, such as the trigger (F) and the response relationship (H); select a subset of behaviors to retarget by clicking the checkbox on each row (E); search behaviors via their natural-language description (D); or create variants of one attribute (I,J). The user can let the system find structurally similar elements to apply the selected behaviors (B) or manually select their own (C).

3.2.1 Step 1: Recording behaviors on existing interfaces. To extract an example behavior, users can follow Umitation’s contextualized instruction (Fig. 3) after initializing the system by clicking its Chrome extension icon. The contextualized instruction design aims to disambiguate user intent when selecting the relevant elements, and it also acts as a memory aid for the use of Umitation. Specifically, to aid the selection, Umitation adds a red dashed border to any HTML element that the users hover their mouse over (Fig. 3 A, B). Users can press the “R” key to select the response object and “T” to select the behavior trigger object. In the dropdown list example, the trigger object is the navigation menu button “Blog,” and the response object is the dropdown list. Once selected, a solid blue border will be added to these elements as selection confirmation (Fig. 3 B). To extract the behavior, a user must initiate the recording (Fig. 3A), hover over the menu button, and wait until the dropdown list’s animated behavior is complete (Fig. 3 C, D). Upon finishing the demonstration, the user stops the recording, and the system will extract the recorded behaviors. This process keeps users’ attention on the visual aspects of the UI, which requires less navigation effort and saves significant time and effort when compared to searching the source code to learn and modify the relevant code—which the users might not be able to find or understand.

During the recording, Umitation tracks the CSS attribute and animation changes on the selected elements (e.g., style, animation) using multiple Web APIs such as MutationObservers (e.g., `MutationObserver.observe(elementNote, config)`), and Element (e.g.,

`elementNote.getAnimations()`).³ Umitation also injects three commonly used event listeners to the trigger elements, including `mouseenter`, `mouseleave`, and `click`. By default, Umitation uses the window object as the trigger object (if the user does not select one), and it tracks multiple window object parameters including page scrolling percentage, window width, and window height. Finally, Umitation stores the timestamps along with all of the changes. Future work can expand the number of attributes and listeners to track additional changes (e.g., `keyup`).

Unlike previous systems such as Theseus [41] or Telescope [23] that help users to discover the implementation of interactive behaviors, Umitation reverse-engineers the observed changes, which bypasses the need to search through multiple function calls across many files. However, it is limited to elements’ visual changes that can be observed from a user demonstration. Additionally, it does not infer any backend implementation, which may lead to inaccurate extraction (e.g., live data that updates the element’s color value). We discuss these limitations in later sections.

3.2.2 Step 2: Viewing and editing behavior details. To make the behaviors easy to understand (DG2), Umitation decomposes each behavior into four distinct components: *trigger*, *trigger object*, *response*, and *response object*. The trigger object and response object can be one or more HTML elements. The trigger and response can be a discrete event, such as a click action or an element’s immediate appearance, or it can be a continuous event, such as a web page

³<https://developer.mozilla.org/en-US/docs/Web/API>

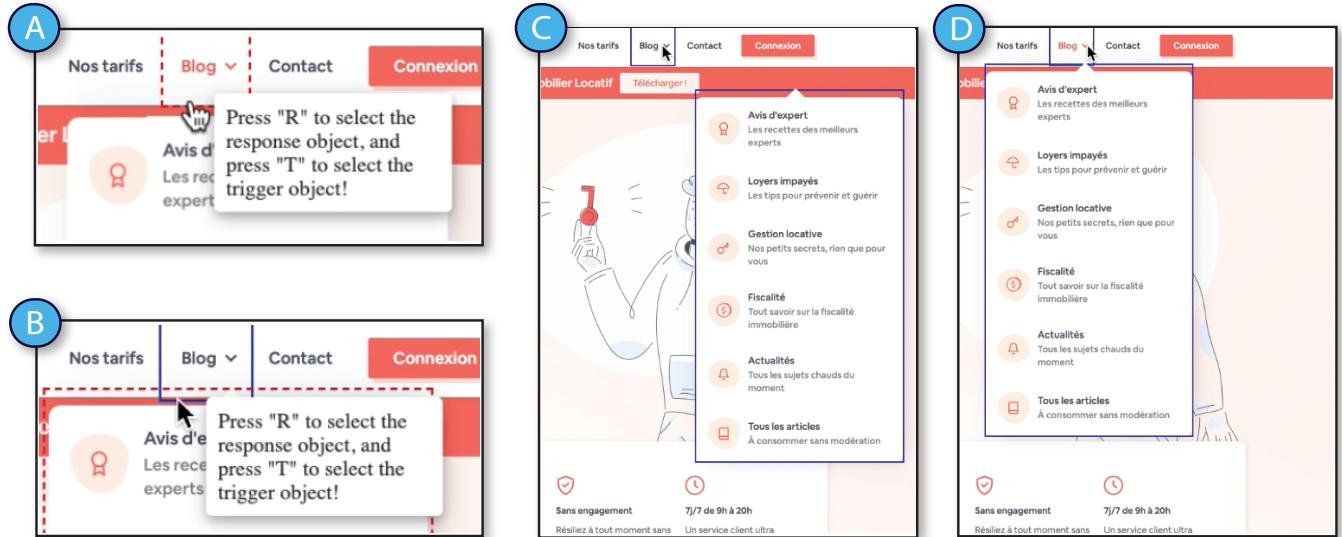


Figure 3: Step 1 of using Umitation. Users follow the instructions on the floating window to select the trigger and response elements (A, B). Then they can record a demonstration of the behaviors that they want to extract (C, D).

scrolling or an element rotating. Furthermore, the relationship between the trigger object and response object can also be dynamic, such as a bar progressing linearly as the page scrolls down. To help users easily analyze the captured behaviors, we have adopted spreadsheet-like UIs from many prior systems [9, 44] and present detailed information about the extracted behaviors in a table, with each row showing one attribute change and each column showing one interaction detail, as explained below:

Select: An index of behaviors that users have selected. It can be used as a reference when retargeting to the target website. A template-based natural-language description of the behavior is also provided (e.g., when [trigger] the [trigger object], then the [response attribute] of [response object] changes [dynamically/immediately]). This is designed to help users easily grasp the high-level details of the behavior and enable the behavior search function (Fig. 2.D). More advanced NLP techniques could be applied in the future to detail the behavior description (e.g., how it changes).

Trigger: The user input event that triggers the behavior. Umitation supports the four most common kinds of user input events, including scrolling, resizing, clicking, and hovering (see Step 1 for implementation details on tracking each). To add expressiveness, Umitation also allows users to switch from one trigger to another from the dropdown menu when applying each behavior. Umitation applies each trigger on the target website using pre-defined JavaScript functions. For instance, if the trigger is scrolling, Umitation will add a `.scroll()` or `.click()` event listener to the trigger object and then add the behavior in the callback functions.

Trigger Object: The HTML tag of the source website element on which the trigger is applied. By default, this object is the window element because two of the triggers—scrolling and resizing—are

commonly applied on the window object. To specify the trigger object, users can change the object from this default when experimenting on the target website.

Response Object: The screenshot of the object on which the behavior happens. Using a third-party script,⁴ Umitation captures the element node screenshots after the user selects them in Step 1. This helps users more easily differentiate between multiple behaviors displayed on the panel. Users must always choose the response object on the target website when experimenting.

Response: Detailed information about the response behavior, including the changed attribute (Fig. 4.H1), the simplified representation of the behavior (Fig. 4.H2), and an attribute relationship chart to show the connection between the changing values of the trigger object and response object or the behavior transition (Fig. 4.H4). The changed attribute shows the associated CSS attribute name, such as `transition`, `width`, or `opacity`. Umitation currently does not support 3D-translated attribute tracking, as it is difficult to visualize this in a 2D chart. Future work can study optimal ways to visualize changes on more CSS dimensions so that users can easily understand and modify them.

The simplified representation of the behavior uses an empty square to represent the response object (Fig. 4.H2). Umitation is designed to visualize two forms of representation: the *start/end* response object states, and the behavior *transition* (i.e., how the response object changes from the *start* state to the *end* state). We use an empty square instead of the response object screenshot because the response object can be arbitrarily complex (e.g., an entire page), which might distract users from the actual response. In addition, if the response behavior is a change in size, the content of a screenshot will be skewed (e.g., the text content will be distorted).

⁴<https://html2canvas.hertzen.com/>

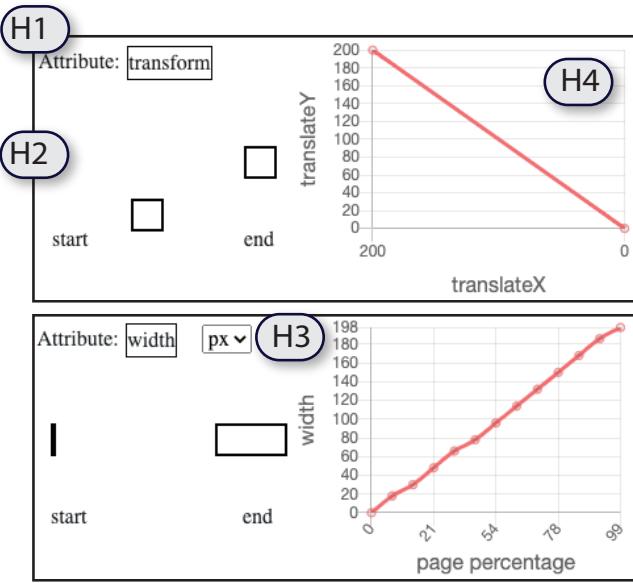


Figure 4: Two visual representations of discrete and continuous UI behavior. 4a is the attribute name. 4b represents the start and end state of the UI. 4c is a dropdown menu that supports unit switching between pixel and %. 4d is a functional relationship chart that shows either the start and end state (top) or the continuous connections between the trigger object and response object.

This design is inspired by Tufte's minimalism theory for effective information visualization [57]. For example, the relationship chart on the first row in Fig. 2.H shows the start state (i.e., translateY and translateX both have values of 200px) and the end state (i.e., translateY and translateX both have values of 0px). To further help users understand this behavior, the relationship chart on the second row plays a looping animation to present an easing function effect.

The design of the attribute relationship chart for the two objects is inspired by prior work in dynamic illustration [26]. Users can also change the extracted relationship for their purposes. Prior work has also shown that creating functional relationships in this way is more intuitive for users without requiring any programming or scripting. Users can create alternative relationships between the objects by using five pre-defined functions (commonly used easing functions⁵) to interpolate the start and end attribute values of the response object: ⇛ reverse, ↘ linear, ↙ ease in, ↖ ease out, ↗ ease in and out, (from left to right, top to bottom), and ⌂ reset the chart to its original. When clicking these buttons, the chart will update its drawing in real time.

One common aspect of UI behavior is that the attribute values will change in % rather than pixels (e.g., in progress bars). However, by default, Umitation captures the pixel values of the following attributes: width, height, left, and top. To consider both cases, Umitation allows users to switch between pixels and % by guiding them to specify the percentage total. When the attribute is one of those

four mentioned above, a dropdown menu appears next to the attribute information (Fig. 4.H3), allowing users to click it to select their preferred unit. Then Umitation will ask users to go back to the source website and select the element for which the width will be used as the percentage total. Once the user selects an element, the chart will update its value to reflect the new calculation (i.e., dividing the original width attribute values by the percentage total and then multiplying by 100).

Others: A list of variants that the user has created for this behavior. This feature is to help designers easily experiment with multiple versions of behaviors of the same attribute (DG3). The design of this feature is guided by work done in the alternative design space where systems can auto-generate a set of similar version instances to provide more design options for users [47, 51]. Users can request that the system auto-generate up to five alternative versions of this behavior. Each new version has a different functional relationship between the response and trigger object. Umitation generates these alternative behaviors using the five pre-defined interpolation methods introduced in the chart manipulation above, in which it applies these functions to the recorded start and end attribute values of the response objects. The generation order (i.e., which version is created first) is also based on the function's popularity: reverse, linear, ease in, ease out, and ease in and out. Users can create variants of one attribute by first clicking the button and then specifying the number of variants they want to create. Each variant is numbered for easier differentiation (i.e., <Behavior_index.variant_cardinal_number>, such as 0.0 for the first behavior and first variant). Once the variants have been created and listed, users can see the relationship reflected in the chart in the "Response" column by clicking the icons next to the behavior's name. More interactive and mixed-initiative approaches can be applied in future work to enhance the system's expressiveness.

3.2.3 Step 3: Experimenting with example behaviors on target websites. To easily experiment with the extracted behaviors on other interfaces (DG3), Umitation supports two methods of retargeting the example behaviors. The first mode is "Manual Select," (Fig. 2.C) in which users will manually specify the target trigger objects and response objects to adapt the behaviors. Similar to Step 1, Umitation provides contextualized instruction to help disambiguate user intent when selecting the target elements. The second mode is "Auto Match," (Fig. 2.B) in which Umitation will automatically find structurally similar elements between the source and target websites to adapt the behaviors. We adapted and modified a matching algorithm [21] that could automatically find similar elements using the differences in four attributes: width, height, top, and left (we labeled these dimensions as obj1 and obj1' in Fig. 5). In contrast to the original algorithm, we used the relative values of these attributes to those of the window object (see details below, e.g., width1 is 18% of the screen, left1 is 20% of the window view port). We use the relative values rather than absolute ones because the scale of the two interfaces might be different when computing their similarity (e.g., when resizing), and because a user's current viewport position relative to the overall HTML page might also be different (e.g. with the original algorithm, a source web page may have fit into one

⁵<https://developer.mozilla.org/en-US/docs/Web/CSS/easing-function>

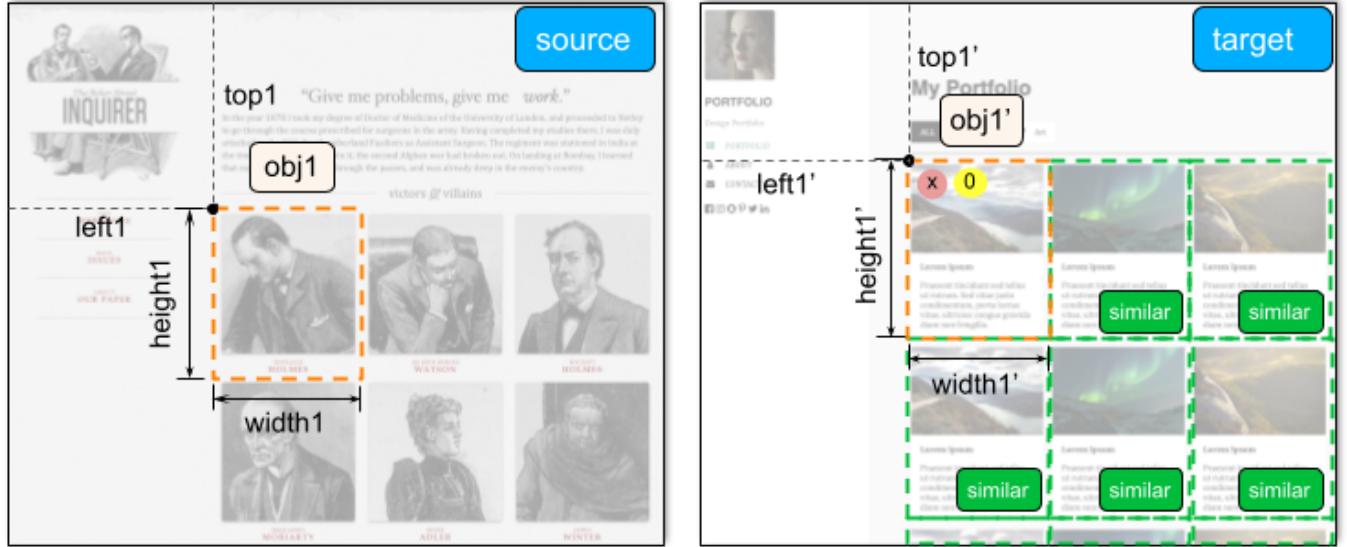


Figure 5: Screenshots of the source (left) and target (right) websites that show the two structurally similar elements, which are $obj1$ and $obj1'$, respectively. Umitation uses four element attributes to compute the difference between objects: left, top, width, and height. Once a similar element is found, a reselection button (red cross) and all the behavior buttons (yellow) will be added to help users experiment more efficiently. Additionally, on the target website, Umitation also searches elements with the same class name (the elements in green dashes) as the chosen one (the element in orange dashes).

view port, but this may not have been the case on the target web page).

Umitation uses a cost function f (eq.1) to compute structural differences between a pair of elements, where obj and obj' are HTML elements in the source and target websites, respectively. $f(obj, obj')$ is the sum of the differences between four object attributes, which is denoted as *attribute*. It consists of the object's relative values—left, top, width, and height—which are computed as shown below:

- left: `element.getBoundingClientRect().x / document.body.clientWidth`
- top: `element.getBoundingClientRect().y / document.body.clientHeight`
- width: `element.clientWidth / document.body.clientWidth`
- height: `element.clientHeight / window.innerHeight`

$$f(obj, obj') = \sum \|obj.attribute - obj'.attribute\| \quad (1)$$

Umitation performs a breadth-first search starting from the body element on the target website to retrieve each object's values. From this point, it computes each element's cost function against the selected ones from the source website (i.e., obj in eq. 1), ranks the elements based on these computations, identifies the elements with the smallest values (i.e., the elements most similar to the element on the source web page), adds a dashed-line box to them (Fig. 5 the orange dashed-line box around $obj1'$), and finally applies the selected behaviors. Note that we did not use the element's type attribute (i.e., `element.tagName`) as one of the factors. This is because we found that 1) many widgets share visual structures but

not types⁶; and 2) developers sometimes define custom elements⁷, making type information less useful.

Additionally, Umitation adds a few buttons to the top-left corner of the element (Fig. 5 the two circled buttons on the top-left corner of $obj1'$) to allow users to cancel and reselect (red X) and to switch the behavior on and off (yellow is on, and gray is off). This design allows users to effortlessly perform in-context “trial and error,” iteratively aligning their design intent and the adapted behaviors. The number on the button matches their behavior index on the main panel. The system also supports group matching, which applies the same behavior to all elements with the same class name (i.e., the five similar card elements in Fig. 5). This design was used because many widgets appear as a group, and so applying a behavior to the whole group helps users infer how they behave as a whole.

Umitation also allows users to apply variants of one behavior to detected similar elements (Fig. 6.B). This design is inspired by prior systems such as Juxtapose [19] and Variolite [27], in which users can easily navigate between versions with browser-tab-like UIs, helping users efficiently compare different behaviors by reducing the effort of navigation and switching between interfaces. Note that when there are more variants than the number of similar elements on the target web page, Umitation will only apply new behaviors to the available elements (i.e., if users apply five variants to Fig. 6, only the first three will be adapted because there are only three elements on which to transfer the new behavior).

⁶Dropdown menu buttons and their lists on these sites are: a) <https://chi2020.acm.org/>: , ; b) <https://www.clickandrent.fr/>: <div>, <nav> (used in study); c) <https://www.figma.com/>: <button>, <div>

⁷<https://angular.io/guide/elements>

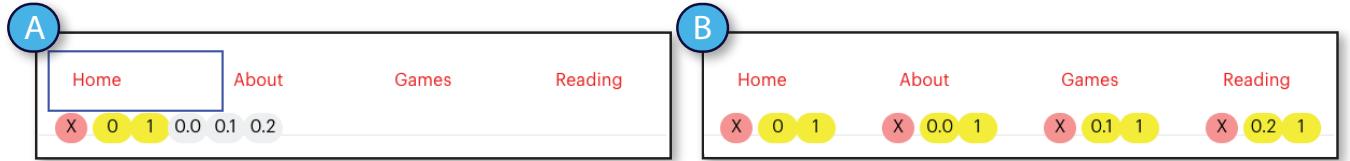


Figure 6: Two forms of experimenting with multiple versions of behavior. A is to experiment all behaviors on one button; B is to experiment each on one button, which helps save the time of toggling each on and off.

In summary, Umitation supports retargeting interactive behaviors with user inputs that are either continuous (e.g., scrolling, resizing, scrubbing) or discrete (e.g., clicking, hovering), and with behaviors in which the changing relationship between the trigger object and response object is dynamic (e.g., easing in and out).

4 SYSTEM EVALUATION

We conducted a remote user study to evaluate Umitation’s usability and efficiency at helping users to extract and retarget UI behavior examples. This evaluation was guided by the usage evaluation in the HCI toolkit evaluation strategy classification [35]. Our goals were to 1) observe and assess how UI designers use Umitation to retarget UI behavior examples from one interface to another, and 2) examine the qualitative usability and utility of Umitation.

A within-subject experiment would have been difficult because there is no clear baseline to compare to Umitation. As we discussed, the most common practices for retargeting UI behaviors are using either professional tools, which do not offer sufficient support for importing and exporting interface assets and therefore cannot handle late-stage UI development; or web programming, which requires participants to know or learn how the task interfaces were implemented. Instead, we asked participants to estimate their workflow times as if they would have to perform the same tasks in our user study with their own tools or systems of choice.

4.1 Participants

We recruited 8 participants (4 women, 4 men, ages 21-38) from both a local participant pool and a freelancer platform. Two of the participants were professional UI designers, and the other 6 were undergraduate or graduate students from multiple schools. All participants had at least one year of UI design and development experience. All of them had experience with CSS/HTML/JS and Chrome DevTools for web interface design; 4 listed professional UI design tools as part of their common practice, including Figma and Adobe XD. Each participant was compensated \$20 for their time.

4.2 Study Design

The remote study session with participants lasted 50-70 minutes. After agreeing to the consent form, each participant first watched a tutorial video of Umitation that showcased the system’s features, and then the participants replicated the example seen in the tutorial. They then performed five tasks (see details in the **Tasks** section below) using Umitation. For each task, the participants first watched a video that included a demonstration and verbal explanation of the example UI behaviors from the source website and the needed behaviors on the target websites. Then they performed the tasks

using Umitation, and the experimenter checked the completion of the task. After the evaluations, the participants reported their demographic information, and we also conducted a short interview with them regarding their experiences with Umitation.

Due to the COVID-19 pandemic, the study was performed remotely using Zoom video conference software. Umitation ran on an experimenter’s laptop. Although it is possible that participants may have experienced minor network delay, we believe the results are still valid for fulfilling the two evaluation goals. The participants were asked to remotely control the laptop using Zoom to perform all of the tasks. They could access the task video at any time, and they could also request the experimenter’s assistance throughout the session. The experimenter’s screen and the interview audio were recorded for each session.

4.3 Tasks

Table 2 shows the details of our user study task information. Because one of our goals when evaluating Umitation was to examine the qualitative usability and utility of the system, we created the study’s five tasks with three further objectives in mind: the UIs should look realistic, the UI behaviors should include critical and low-level details that are often omitted when specifying interactive behaviors in practice [38], and the tasks should help to demonstrate Umitation’s full capability, which existing tools cannot easily do (in terms of complexity and diversity) [48]. To achieve these objectives, we used two professional websites (Clickandrent⁸ and BakerStreet⁹) to be the source websites and two additional websites (Webflow¹⁰ and Portfolio¹¹) to be the target websites. We modified their UIs to result in behaviors that are highly dynamic and type-diverse: multiple elements that respond to user input, multiple attribute changes on one element, user inputs that are either continuous (e.g. resizing) or discrete (e.g., mouse hover), and attribute value changes that can be either relative or absolute (see Supplemental Materials for screenshots of the task interfaces).

4.4 Results

4.4.1 Time and Accuracy. Of the 40 scenario instances (8 participants x 5 tasks), the participants successfully completed all of them. Table 2 lists the average time (in seconds) each participant spent working and the number of times they received assistance across all five tasks. Of these forms of assistance, more than half of them were

⁸<https://tinyurl.com/3uj2yvux>

⁹<https://tinyurl.com/ewun2pmh>

¹⁰<https://tinyurl.com/3zzjtzdd>

¹¹<https://tinyurl.com/6r822vvx>

#	Task Description	Example Web UI	Use of Umitation
1	Click the button, and the page scrolls to the top with an animated effect.	A long list of items or members to display (e.g. www.uist.acm.org/uist2021/organizers.html).	Help selecting the desired button behaviors (one button can have multiple behaviors associated with the clicking event).
2	Resize the window horizontally, and the size of a set of similar elements changes together continuously.	Similar items such as shopping items have the same size-changing responsive effect when resizing the window (e.g., ebay.com).	Help specifying desired elements by automatically finding structurally similar elements during Step 1 (selecting elements to extract behavior) and Step 3 (selecting elements to adapt behaviors).
3	Resize the window horizontally, and the sizes of multiple different elements change continuously.	Multiple different elements such as the side bar, navigation bar, and items have different responsive effects when resizing the window (e.g., nytimes.com).	Help specifying multiple desired elements by automatically finding multiple structurally similar elements during Step 3 (selecting elements to adapt behaviors).
4	Hover over the navigation menu, and the dropdown menu appears with an animated effect.	Experimenting with different animated effects on the dropdown menu list's appearance (e.g., youtu.be/AIdslaUj9wg?t=1270).	Help testing various behavior appearances of different dropdown menus found on the target websites.
5	Scroll down the page, and a progress bar changes continuously.	A fixed-position progress bar that changes its percentage accordingly as users scroll the page (e.g., the progress bar on any paper page on https://dl.acm.org/conference/uist).	Help with identifying and specifying the units of the attribute values in the extracted behavior, and selecting the elements accordingly (e.g., the progress bar or the progress bar background).

Table 1: The five UI behavior tasks users retargeted in the user study. First column: task index; second column: the task description we gave to participants in our study; third column: example UI behaviors that were used on existing professional websites; fourth column: their corresponding use with Umitation.

requests for memory aid (e.g., the participants jumped ahead or forgot the next steps of the workflow); two participants did not quite understand the rationale behind the percentage (relative value) vs. the pixel (absolute value). When asked to reflect on their experience, the participants all mentioned that once they became familiar with the tool later in the session, they could easily perform all the tasks. We also asked the participants to report the estimated time they would have spent on the same tasks with their current practices (i.e., without using Umitation). Three participants reported days (P2, P4, P8), three reported hours (P1, P3, P5), and two reported 30 minutes or less (P6, P7).

4.4.2 Usability and Efficiency. After the study, the participants rated Umitation’s ease of use and efficiency on a seven-point Likert scale from “strongly disagree” to “strongly agree.” Umitation scored on average 5.7 (SD = 0.92) on “I found Umitation ease to use.” Participants reported that they found Umitation to be “easy to use” (P1, P3, P6) and “easy to understand” (P5, P7), and that they found that the real-time guidance, such as the floating instruction window (Fig. 3 A, B), made the workflow straightforward (P6). “For the Auto Match feature, the first time I used it, it was exactly how I imagined it would be” (P5). Umitation scored on average 6.1 (SD=0.3) on “Umitation is efficient in helping me with retargeting UI behavior examples compared to existing tools I’ve used before,” and participants found it to be “very useful” (P5).

In summary, participants found Umitation to be a “powerful tool” (P5) with a low learning threshold to users but high expressiveness (P2), and they stated that it was “*really good at interaction (behavior) type of tasks*” (P7). In particular, we found that Umitation helped designers to easily extract any UI behavior examples by enabling them to directly interact with the interfaces to demonstrate the desired behaviors, to understand complex UI behavior by unpacking the behavior via tables, and to intelligently guide users to specify their intent by automatically matching and supporting variant comparisons. In the followup interviews, participants reported their thought process while using Umitation and how they compared Umitation to their current practices.

4.4.3 Learning Effort. Compared to their current practices, all participants stated that Umitation made the example behavior learning process much easier. “*The biggest advantage is it’s very visual and non code-base[d]. If I’m just a designer, and I don’t know anything about the coding part, it’s very user-friendly, where you see the behavior[s] that you want, and then you copy and paste them*” (P4). Learning UI behaviors on existing websites can be burdensome for designers, who must inspect the relevant source code to locate and understand the implementations. Not only does this process shift their attention away from the visual and dynamic aspects of the behavior itself, but understanding other developers’ code,

Task index	Completion time (Avg. in sec. (S.D.))	# of assistance
1	114.3 (63.2)	2.0 (1.3)
2	209.5 (88.9)	1.0 (1.1)
3	74.4 (35.0)	0.9 (0.8)
4	208.4 (152.4)	0.9 (0.6)
5	185.4 (139.0)	0.5 (0.8)

Table 2: Average time spent (sec.) on each task and the number of times that participants received system usage hints from the experimenter.

Question	7-point Likert scale response (Avg. (S.D.))
Umitation is easy to use	5.1 (0.8)
Umitation is efficient at helping me complete the tasks	5.8 (1.0)

Table 3: Average Likert scale responses by the eight participants for two questions regarding Umitation’s usability and efficiency.

which might not be well-formatted or explained, can be a complex task [58].

Umitation removes these burdens by enabling designers to record and reuse what they see without having them to dig into the source code. Similar to prior work that argues that element-based animations should be treated as first-class objects [11], Umitation treats each UI behavior as a first-class object, where the low-level details such as triggers and responses are its attributes. This allows designers to focus on only the visual and interactive aspects of the behaviors, making the extraction process easier.

4.4.4 Behavior Understanding. Similar to the clipboard feature on any modern text editor, the extracted behaviors are saved temporarily on Umitation’s control panel. However, unlike the traditional content copy-paste techniques where users would know exactly what was copied to their clipboard, the complex UI behaviors often consist of multiple attribute changes. In our study, we found that Umitation helped participants to clearly see and understand these changes in detail. “*I was expecting maybe like one behavior, but there ended up to be four. So there are some ones that maybe I didn’t even notice [were] present in the original source*” (P6). Systems like Telescope can help easily find relevant code, but for complex UI behaviors, simply having the code might not be enough for users to quickly grasp the relationship between different components within the behaviors. “*The visual representation of it is much nicer, because you can immediately see [that] this is the linear process for this example. I think if it’s just written in code, it can be confusing to visualize sometimes, but because you’ve got that graph, it’s much easier*” (P3). Additionally, Umitation helped designers to learn about the extracted behaviors by making them editable and interactable on its system panel. “*I really like how things are presented in columns where I can scan through each of them to compare and find the information on the same category*” (P4). Furthermore, via manipulation of the details of the behavior, Umitation helped designers to easily explore and express their design ideas (e.g., create variants) rather than relying on the exact same behaviors that were in the source website, making the design process more expressive and scalable.

4.4.5 Scalability. An important step in the UI design process is to compare different versions of a design to see which one fits better in context [50]. By adding the multi-version tab buttons on the target elements, Umitation eased the process of switching between versions, helping participants make decisions more efficiently. “*I would personally use it to help make design decisions. Or [when] there are multiple ways in which I’d want a website to behave and I can’t decide between them*” (P3). Additionally, with the structurally similar element matching technique, Umitation intelligently guided users to specify and disambiguate their intentions, which limited the effort of manually choosing which elements to extract and match.

4.4.6 Team Productivity. Half of the participants also noted an additional benefit: Umitation can help users communicate UI designs to others, allowing users to get feedback or showcase their ideas in a contextualized way. “*Communicating verbally or pointing to reference websites might be less effective than this tool*” (P2). As prior work has shown [12, 13, 32], presenting contextualized visual information helps people ground communication. “*When you’re trying to explain these kinds of things to other people, like developers, for instance, or your designer, you might point to other websites, and like [say] ‘please do something like that.’ And then it might not work or [the result] might be quite different from [actuality] in this context*” (P8).

5 DISCUSSION

The study results have shown Umitation’s promise and effectiveness in helping designers experiment with example UI behaviors within minutes. By offering contextualized guidance to help users specify coding details, Umitation allows designers to avoid the effort of implementing complex behaviors themselves, thus providing a low barrier to entry. Essentially, Umitation provides what David Kelley terms “enlightened trial and error,” in which prototypes can be more fluidly produced and evaluated. We now discuss Umitation’s role in the design lifecycle, system limitations, and areas for future research.

5.1 Umitation's Role in the Design Lifecycle

Our user study has demonstrated Umitation's strength in supporting retargeting complex and dynamic UI behaviors from one interface to another, but participants also mentioned that Umitation can be used at the initial stage of the design process, as referring to existing designs found in the wild is a common practice throughout the design lifecycle. Unlike the basic, single function copy-and-paste tool found in many devices, Umitation allows designers to collect, store, manage, and retrieve (Fig. 2.D) example behaviors, serving as an example library for future usage or sharing with others. Additionally, the extracted example behaviors will continue to work even if the source website changes.

5.2 System Scope and Study Limitations

Umitation was designed to capture/adapt “front end” visual behaviors that occur within the DOM tree of a Website. It was not designed to and cannot capture a behavior’s underlying computational functions. For instance, UM could cover the visualization of password strength, but not the functionality that assesses its strength (2AC). Umitation is most effective at retargeting dynamic UI behaviors that involve dynamic transformation across multiple CSS attributes of multiple interface elements. It is less necessary for simple attribute changes, such as width or color changes that occur after clicking a button, as this relevant code would be easy to find. Our matching algorithm could be improved to cover more complex cases. For instance, an earlier version of Umitation had the ability to search for matches within a user-specified region of the target site, although our initial pilot studies suggested that manual selection would suffice. Our system evaluation is also limited because no baseline was used for comparison. In addition, the tasks used in the study were not from the real world. Ideally, we would want the participants to use Umitation for their own design projects.

Our study also revealed multiple system limitations. First, when specifying elements on the source website (Step 1), Umitation requires users to manually select accurate elements that will respond to their input. If users incorrectly select specific elements, Umitation does not provide any feedback or guidance to help correct them. For instance, one participant (P4) first selected the entire source page in one task and wanted to avoid the low-level “tedious” process of finding the correct element from which to extract behaviors. Second, when first reading and analyzing the behaviors (Step 2), participants felt “overwhelmed” (P5, P6) by the amount of information presented in the table. Their initial feeling of being overwhelmed might have also resulted from the Umitation’s unfamiliar visual representations, which might have been new to designers who were more familiar with the traditional canvas-editor-based representation. Third, when applying the behaviors to the target website (Step 3), Umitation requires the target elements to be visible such that they can be used as target objects for the behaviors. Furthermore, Umitation cannot display accurate example behaviors on the target elements if there are other effects or constraints that cannot be overwritten. In other words, Umitation cannot combine the extracted behaviors with the existing ones on the same attributes; rather, it overwrites the original behaviors.

5.3 Future Work

Participants provided suggestions on improving Umitation’s efficacy and usability. Two participants were curious about ways they could implement the example behaviors after retargeting them via Umitation. While Umitation does not trace the source code of the original implementation, future work can display the synthesized code snippets that generated the retargeting effects, which prior work has shown to help users implement the behaviors in their own codebase [8]. Although Umitation cannot capture a behavior’s underlying computational functions, prior work such as Fusion [60] has already presented techniques to support that function. Future work could combine these systems to support a wider range of UI retargeting tasks than either could alone. We also plan to explore techniques in the program synthesis literature [28, 59] to help users to efficiently integrate the code snippets of the extracted behaviors into the source code of the target website.

Additionally, we observed that when choosing elements from which to extract or adapt example behaviors, three participants first selected those that contained other unintended elements (e.g., the entire body element). One participant expressed that “*it would be really nice if you could extract and adapt the behaviors of the entire website*” (P4). This preference makes sense because identifying the exact elements can be tedious. Similar to Bricolage [33], future systems could design models to predict the intended behavior match without relying on a user’s manual specification. Because Umitation is still a research prototype, it has not yet been engineered with robust system recovery mechanisms. During the user study, participants would restart Umitation when encountering unexpected system states (e.g., after capturing the entire source page’s behaviors). Future work could add guidance to help users repair system breakdowns, or introduce an error message system to help users better understand the system states.

6 CONCLUSION

In this paper, we introduced an easy and expressive approach for UI designers to retarget example behaviors found in the wild to their own interfaces. We designed and developed Umitation, which instantiates this approach, to enable users to extract example UI behaviors, modify and remix them, and retarget them to different interfaces. Participants found Umitation powerful and intuitive and expressed its high efficiency compared to their existing practices, such as programming. Umitation is a first step in helping disambiguate user intent during the functional copy-and-paste process by providing a natural and expressive method of retargeting dynamic interface behaviors.

7 ACKNOWLEDGEMENTS

This research was supported in part by the National Sciences and Engineering Research Council of Canada (NSERC) under Grant IRCPJ 545100 - 18.

REFERENCES

- [1] 2021. Microsoft Format Painter. <https://tinyurl.com/afxc85n9> Accessed: April, 2021.
- [2] 2021. Scrimba. <https://www.scrimba.com/> Accessed: April, 2021.
- [3] Margaret A Boden et al. 2004. *The creative mind: Myths and mechanisms*. Psychology Press.

- [4] Elizabeth Boling and Theodore W Frick. 1997. Holistic rapid prototyping for web design: Early usability testing is essential. *Web-based instruction* (1997), 319–328.
- [5] Brian Burg, Richard Bailey, Amy J Ko, and Michael D Ernst. 2013. Interactive record/replay for web application debugging. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 473–484.
- [6] Bill Buxton. 2010. *Sketching user experiences: getting the design right and the right design*. Morgan kaufmann.
- [7] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. 2019. Everybody dance now. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 5933–5942.
- [8] Kerry Shih-Ping Chang and Brad A Myers. 2012. WebCrystal: understanding and reusing examples in web authoring. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3205–3214.
- [9] Kerry Shih-Ping Chang and Brad A Myers. 2014. Creating interactive web data applications with spreadsheets. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 87–96.
- [10] Sarah E Chasins, Maria Mueller, and Rastislav Bodik. 2018. Rousillon: Scrapping Distributed Hierarchical Web Data. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 963–975.
- [11] Yan Chen, Sang Won Lee, and Steve Oney. 2021. CoCapture: Effectively Communicating UI Behaviors on Existing Websites by Demonstrating and Remixing. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*.
- [12] Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S Lasecki, and Steve Oney. 2017. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. 6220–6231.
- [13] Yan Chen, Steve Oney, and Walter S Lasecki. 2016. Towards providing on-demand expert support for software developers. In *Proceedings of the 2016 CHI conference on human factors in computing systems*. 3192–3203.
- [14] Pei-Yu Chi, Sen-Po Hu, and Yang Li. 2018. Doppio: Tracking ui flows and code changes for app development. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–13.
- [15] Allen Cypher and Daniel Conrad Halbert. 1993. *Watch what I do: programming by demonstration*. MIT press.
- [16] Steven Dow, Julie Fortuna, Dan Schwartz, Beth Altringer, Daniel Schwartz, and Scott Klemmer. 2011. Prototyping dynamics: sharing multiple designs improves exploration, group rapport, and results. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2807–2816.
- [17] Mary L Gick and Keith J Holyoak. 1983. Schema induction and analogical transfer. *Cognitive psychology* 15, 1 (1983), 1–38.
- [18] Tovi Grossman, Justin Matejka, and George Fitzmaurice. 2010. Chronicle: capture, exploration, and playback of document workflow histories. In *Proceedings of the 23rd annual ACM symposium on User interface software and technology*. 143–152.
- [19] Björn Hartmann, Leslie Wu, Kevin Collins, and Scott R Klemmer. 2007. Programming by a sample: rapidly creating web applications with d. mix. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 241–250.
- [20] Björn Hartmann, Loren Yu, Abel Allison, Yeonsoo Yang, and Scott R Klemmer. 2008. Design as exploration: creating interface alternatives through parallel authoring and runtime tuning. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*. 91–100.
- [21] Yasunari Hashimoto and Takeo Igarashi. 2005. Retrieving Web Page Layouts using Sketches to Support Example-based Web Design.. In *SBM*. Citeseer, 155–164.
- [22] Scarlett R Herring, Chia-Chen Chang, Jesse Krantzler, and Brian P Bailey. 2009. Getting inspired! Understanding how and why examples are used in creative design practice. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 87–96.
- [23] Joshua Hibscher and Haoqi Zhang. 2015. Unravel: Rapid web application reverse engineering via interaction recording, source tracing, and library detection. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 270–279.
- [24] Joshua Hibscher and Haoqi Zhang. 2016. Telescope: Fine-tuned discovery of interactive web UI feature implementation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*. 233–245.
- [25] Yue Jiang, Wolfgang Stuerzlinger, and Christof Lutteroth. 2021. ReverseORC: Reverse Engineering of Resizable User Interface Layouts with OR-Constraints. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–18.
- [26] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: sketching dynamic and interactive illustrations. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. 395–405.
- [27] Mary Beth Kery, Amber Horvath, and Brad A Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists.. In *CHI*, Vol. 10. 3025453–3025626.
- [28] Mary Beth Kery, Donghao Ren, Fred Hohman, Dominik Moritz, Kanit Wong-suphasawat, and Kayur Patel. 2020. mage: Fluid Moves Between Code and Graphical Work in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 140–151.
- [29] Donghwi Kim, Sooyoung Park, Jihoon Ko, Steven Y Ko, and Sung-Ju Lee. 2019. X-Droid: A Quick and Easy Android Prototyping Framework with a Single-App Illusion. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 95–108.
- [30] Huhn Kim and Wan Chul Yoon. 2005. Supporting the cognitive process of user interface design with reusable design cases. *International journal of human-computer studies* 62, 4 (2005), 457–486.
- [31] Janet L Kolodner and Linda M Wills. 1993. Case-based creative design. *AISB QUARTERLY* 85 (1993), 1–8.
- [32] Robert E Kraut, Susan R Fussell, and Jane Siegel. 2003. Visual information as a conversational resource in collaborative physical tasks. *Human-computer interaction* 18, 1-2 (2003), 13–49.
- [33] Ranjitha Kumar, Jerry O Talton, Salman Ahmad, and Scott R Klemmer. 2011. Bricolage: example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2197–2206.
- [34] James A Landay and Brad A Myers. 1995. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 43–50.
- [35] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. 2018. Evaluation strategies for HCI toolkit research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–17.
- [36] Brian Lee, Savil Srivastava, Ranjitha Kumar, Ronen Brafman, and Scott R Klemmer. 2010. Designing with interactive example galleries. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2257–2266.
- [37] Germán Leiva and Michel Beaudouin-Lafon. 2018. Montage: A Video Prototyping System to Reduce Re-Shooting and Increase Re-Usability. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 675–682.
- [38] Germán Leiva, Nolwenn Maudet, Wendy Mackay, and Michel Beaudouin-Lafon. 2019. Enact: Reducing designer-developer breakdowns when prototyping custom interactions. *ACM Transactions on Computer-Human Interaction (TOCHI)* 26, 3 (2019), 1–48.
- [39] Germán Leiva, Cuong Nguyen, Rubaiat Habib Kazi, and Paul Asente. 2020. Pronto: Rapid Augmented Reality Video Prototyping Using Sketches and Enaction. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) (CHI '20). Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376160>
- [40] Toby Jia-Jun Li, Amos Azaria, and Brad A Myers. 2017. SUGILITE: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. 6038–6049.
- [41] Tom Lieber, Joel R Brandt, and Rob C Miller. 2014. Addressing misconceptions about code with always-on programming visualizations. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2481–2490.
- [42] Sarah Lim, Joshua Hibscher, Haoqi Zhang, and Eleanor O'Rourke. 2018. Ply: A visual web inspector for learning from professional webpages. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 991–1002.
- [43] James Lin, Mark W Newman, Jason I Hong, and James A Landay. 2000. DENIM: finding a tighter fit between tools and practice for Web site design. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. 510–517.
- [44] James Lin, Jeffrey Wong, Jeffrey Nichols, Allen Cypher, and Tessa A Lau. 2009. End-user programming of mashups with vegemite. In *Proceedings of the 14th international conference on Intelligent user interfaces*. 97–106.
- [45] Michael Xieyang Liu, Jane Hsieh, Nathan Hahn, Angelina Zhou, Emily Deng, Shaun Burley, Cynthia Taylor, Aniket Kittur, and Brad A Myers. 2019. Únakeit: Scaffolding Developers' Decision-Making Using the Web. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*. 67–80.
- [46] Josip Maras, Maja Stula, Jan Carlson, and Ivica Crnkovic. 2013. Identifying code of individual features in client-side web applications. *IEEE Transactions on Software Engineering* 39, 12 (2013), 1680–1697.
- [47] Justin Matejka, Michael Glueck, Erin Bradner, Ali Hashemi, Tovi Grossman, and George Fitzmaurice. 2018. Dream lens: Exploration and visualization of large-scale generative design datasets. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.
- [48] Brad Myers, Sun Young Park, Yoko Nakano, Greg Mueller, and Amy Ko. 2008. How designers design and program interactive behaviors. In *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*. IEEE, 177–184.
- [49] Brad A Myers, Ashley Lai, Tam Minh Le, YoungSeok Yoon, Andrew Faulring, and Joel Brandt. 2015. Selective undo support for painting applications. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. 4227–4236.
- [50] Mark W Newman and James A Landay. 2000. Sitemaps, storyboards, and specifications: A sketch of web site design practice. In *Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques*. 263–274.
- [51] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2015. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. 1221–1224.
- [52] Stephen Oney and Brad Myers. 2009. FireCrystal: Understanding interactive behaviors in dynamic web pages. In *2009 IEEE Symposium on Visual Languages*

- and Human-Centric Computing (VL/HCC)*. IEEE, 105–108.
- [53] Jungkook Park, Yeong Hoon Park, and Alice Oh. 2018. Non-Linear Editing of Text-Based Screencasts. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 403–410.
 - [54] Jeffrey Stylos, Brad A Myers, and Andrew Faulring. 2004. Citrine: providing intelligent copy-and-paste. In *Proceedings of the 17th annual ACM symposium on User interface software and technology*. 185–188.
 - [55] Amanda Swearngin, Chenglong Wang, Alannah Oleson, James Fogarty, and Amy J Ko. 2020. Scout: Rapid Exploration of Interface Layout Alternatives through High-Level Design Constraints. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
 - [56] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomir Měch. 2012. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. 63–74.
 - [57] Edward R Tufte. 1983. *The visual display of quantitative information*. Vol. 2.
 - [58] April Yi Wang, Zihan Wu, Christopher Brooks, and Steve Oney. 2020. Callisto: Capturing the "Why" by Connecting Conversations with Computational Narratives. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.
 - [59] Yifan Wu, Joseph M Hellerstein, and Arvind Satyanarayanan. 2020. B2: Bridging Code and Interactive Visualization in Computational Notebooks. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. 152–165.
 - [60] Xiong Zhang and Philip J Guo. 2018. Fusion: Opportunistic web prototyping with ui mashups. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 951–962.
 - [61] Nanxuan Zhao, Nam Wook Kim, Laura Mariah Herman, Hanspeter Pfister, Rynson WH Lau, Jose Echevarria, and Zoya Bylinskii. 2020. Iconate: Automatic compound icon generation and ideation. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–13.