Individualising Graphical Layouts with Predictive Visual Search Models

KASHYAP TODI and JUSSI JOKINEN, Department of Communication and Networking, Aalto University, Finland
KRIS LUYTEN, UHasselt - tUL - Flanders Make, Belgium
ANTTI OULASVIRTA, Department of Communication and Networking, Aalto University, Finland

In domains where users are exposed to large variations in visuo-spatial features among designs, they often spend excess time searching for common elements (features) on an interface. This article contributes individualised predictive models of visual search, and a computational approach to restructure graphical layouts for an individual user such that features on a new, unvisited interface can be found quicker. It explores four technical principles inspired by the human visual system (HVS) to predict expected positions of features and create individualised layout templates: (I) the interface with highest frequency is chosen as the template; (II) the interface with highest predicted recall probability (serial position curve) is chosen as the template; (III) the most probable locations for features across interfaces are chosen (visual statistical learning) to generate the template; (IV) based on a generative cognitive model, the most likely visual search locations for features are chosen (visual sampling modelling) to generate the template. Given a history of previously seen interfaces, we restructure the spatial layout of a new (unseen) interface with the goal of making its features more easily findable. The four HVS principles are implemented in Familiariser, a web browser that automatically restructures webpage layouts based on the visual history of the user. Evaluation of Familiariser (using visual statistical learning) with users provides first evidence that our approach reduces visual search time by over 10%, and number of eye-gaze fixations by over 20%, during web browsing tasks.

CCS Concepts: • **Human-centered computing → Human computer interaction (HCI)**; **User interface design**;

Additional Key Words and Phrases: Visual search, graphical layouts, computational design, adaptive user interfaces

ACM Reference format:
Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2019. Individualising Graphical Layouts with Predictive Visual Search Models. *ACM Trans. Interact. Intell. Syst.* 10, 1, Article 9 (August 2019), 24 pages.
https://doi.org/10.1145/3241381

This article is an extension of "Familiarisation: Restructuring Interfaces with Visual Learning Models" [41].
The reviewing of this article was managed by special issue associate editors Mark Billinghurst, Margaret Burnett, and Aaron Quigley.
The project has partially received funding from the Academy of Finland project COMPUTED and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 637991).
Authors' addresses: K. Todi, J. Jokinen, and A. Oulasvirta, Department of Communication and Networking, Aalto University, PO Box 11000, Helsinki, Finland; emails: kashyap.todi@gmail.com, {jussi.jokinen, antti.oulasvirta}@aalto.fi; K. Luyten, UHasselt - tUL - Flanders Make, Wetenschapspark 2, Diepenbeek, Belgium; email: kris.luyten@uhasselt.be.

2160-6455/2019/08-ART9 $15.00
https://doi.org/10.1145/3241381

1 INTRODUCTION

This article addresses a common predicament in interaction with graphical user interfaces: from blogs to banking and mobile apps, users encounter a wide diversity of visual designs. Even when serving the same purpose, designs differ in terms of element positions, colouring, images, and widget types. While visual uniqueness provides for identity and brand recognition, users are constantly confronted with learning, relearning, and accustoming to navigate new structures and different styles. For the visual system, this poses a challenge to constantly adapt the use of visual attention. By understanding and modelling how we visually learn designs as and when they are encountered, we could design and adapt interfaces automatically such that elements can be more easily found. This could make interface ecologies more usable and enjoyable for users who care less about aspects such as brand identity.

Our work is motivated by the observation that the added effort experienced upon encountering unfamiliar or atypical designs depends on the cost related to visual search for new and unfamiliar visual layouts [8, 21, 40]. Consider taking a familiar design and moving an element to a different position. Users would first seek the element in its expected place and, upon failing, continue with some search strategy to locate the element that was moved, or simply give up. However, designs are likely to differ in more than one feature, requiring guessing of element positions, and further frustration.

This article investigates individualised models of visual search to predict expected positions of features on an interface, and applies it to restructuring new layouts. Our goal is to predict where users assume elements to be on interfaces they have not seen before. More formally, our goal can be defined as follows:

> *Given a new design d, previously unseen by the user, and a history of previously visited visual designs H ($d \notin H$), restructure d to minimise costs to visual search for the user.*

By *restructuring*, we refer to manipulations to the visio-spatial layout, such as moving, resizing, and recolouring elements. There are multiple competing principles on which the visual search model can be based, depending on how it is defined. As these principles can lead to different techniques of restructuring, it is important to explore their assumptions, implementations, and results. To this end, we define and formalise four modelling principles, informed by theories of human visual system (HVS) and visual learning, illustrated in Figure 1. We follow a white-box approach, where the models come from research on human memory and perception. The benefits of this over data-driven approaches, wherein a predictive model of attention would be learned from data, are two-fold: (1) parameter estimation: because of a low number of parameters, we can train the model to an individual with fewer observations; (2) transparency: we can fully expose the operating principle of the method.

These principles are studied and instantiated here in a system called *Familiariser*, which restructures new graphical layouts such that they are more familiar to the user. To achieve this, three main components are required: history logging, visual search modelling, and interface restructuring (Figure 2). Familiarisation exploits the capability of operating systems and browsers to change an interface design dynamically.

At the user-end, familiarisation exploits known processes of human visual system in visual search. This ensures that familiarised layouts are more usable due to predictable element locations and layout structures. Such computer-driven familiarisation complements efforts at design-time to ensure *consistency* and adherence to design standards and guidelines [30, 36]. While consistency enforced at design time can only approximate what the user population has experienced, ensuring

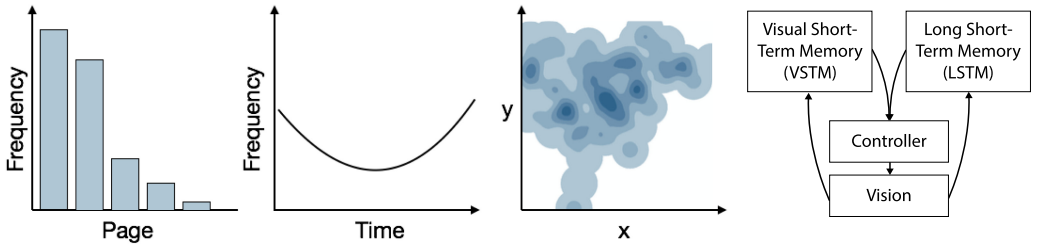


Fig. 1. An illustration of the four principles for modelling visual search, and creating an individualised template. (I) selects the page with highest frequency; (II) considers frequency and time, selecting the page with the highest value on the curve; (III) considers positional (xy) information, and creates a probability distribution of features; (IV) uses a cognitive model of visual search to determine feature locations.

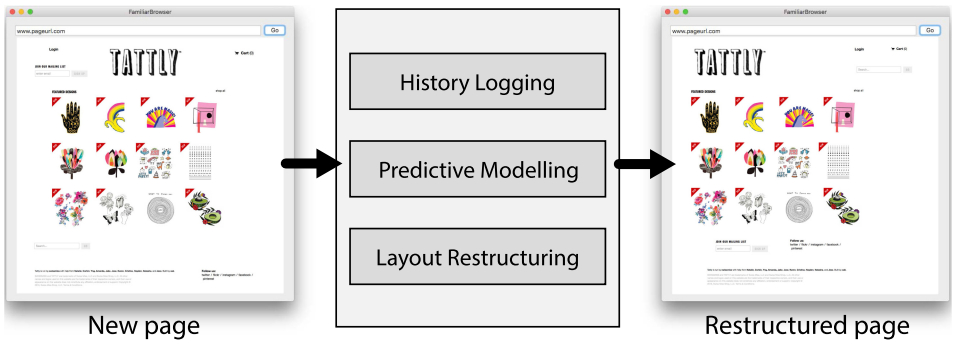


Fig. 2. Main components for individualised restructuring of an interface. As users visit interfaces, their history is logged. This is used to model visual search and generate a template for the user. When a new interface is visited, it applies this template to restructure the page.

consistency through active familiarisation at runtime allows, in principle, perfect consistency for an individual. *Familiariser* is a browser-based system that dynamically restructures website layouts to make them easier to use. Results from our study indicate that familiarisation can reduce visual search times by over 10%, and also leads to a reduction of 23% in eye fixations when searching for elements.

Overview: The Four Principles for Modelling Visual Search

The set of restructuring techniques we examine here can be generally defined as functions that take as input user history H and a new design d, and yield a restructured design d': $f(d, H) \rightarrow d'$. In this article, we explore four principles (Figure 1) with different theoretical underpinnings and consequences to how restructuring operates and how they influence results. Section 3 *"Principles for Individualised Modelling"* elaborates on each of these principles in greater detail.

I. Frequency: The most straightforward approach is to simply take the most *frequently used design* from H. Any new design within the same domain would be restructured such that elements appear in approximately the same places. While this design is likely to be recognizable, and simple to implement, the approach does not take into account the fact that most recent experiences are likely to dominate recall. Something that might have been frequently used in the past might be partially forgotten and overshadowed by the most recently encountered designs.

II. Serial Position Curve: The second principle implements the well-known serial position function of long-term recall [11]. Extensive empirical research on long-term and short-term memory has shown that the first and the most recently encountered objects are better recalled than ones in the middle. We implement a mathematical function that describes the relationship between order and recall probability. This is used to choose the most-likely-to-be-recalled design in H according to this function. A limitation of this and the first approach is that the visual features might not be adequately described by a single design.

III. Visual Statistical Learning: The third principle is visual statistical learning (VSL), according to which visual attention relies on a probabilistic internal model sensitive to the characteristics of the environments they have encountered before [8]. We implement this hypothesis by building a statistical model of visual features in H. It is used to estimate the "most probable" location of a given element. We then restructure a layout, element by element, considering this function, and finally align the elements so that they appear orderly.

IV. Cognitive Model: In our final technique, we train a cognitive model of layout learning with H and use its prediction on an empty canvas to predict the most likely positions of the elements of d. The model simulates learning of visual positions, and generates visual search patterns and times, given a layout and knowledge on that layout [21]. As an input it requires H and associated visitation durations, and optionally also considers element frequencies or relevance of elements. In this, it is similar to the previous approach. However, the model generates the actual eye movement patterns and visual search times, in addition to generating the memorability of location elements. It also simulates forgetting of layouts that have been visited only shortly and further back in time. This allows the model to be used for evaluating automatically how the familiarised layout will impact the user's behaviour, as well as how the user learns the new layout.

2 RELATED WORK

Our work is situated within the domains of (1) visual search modelling, (2) layout generation and restructuring, and (3) user interface adaptation. We discuss prior literature that has dealt with these themes, and draw comparisons to our work.

2.1 Visual Search

Visual search requires the user to scan through a *search array* to find a target [35]. This process is shaped by the limitations of the human vision and information processing. The area of clear visual acuity (*fovea*) is limited, and therefore in the case of UIs, the user is able to see only partially its graphical elements at any given time. In visual search, the human visual system is therefore faced with an attention deployment problem: what visual element to attend next?

Attention deployment during visual search has been modelled using various computational models [23]. A popular computational search model computes salience of regions in an image, and uses that to predict what the most probably attended regions will be [19]. In also implements an *inhibition of return*, where already searched regions are not revisited. This search model can be called *bottom up* model, because its attention deployment computation is based on features of the visual field. Another approach, *top down*, emphasises the role of the search task. In the case of UIs, the user probably has some task-relevant information, which can be used to guide the search. For instance, a model of visual search on keyboard layouts utilised this kind of approach, modelling the impact of learning on visual search [21]. In this work, we target graphical layouts, where users apply visual search to find elements of interest on a screen. By constructing models of a user's visual search strategy, we can automatically construct or adapt layouts that might reduce visual search.

2.2 Layout Generation and Interface Restructuring

Automatic generation of layouts, and restructuring existing designs, have received significant attention by researchers and practitioners. One of the main rationales behind automatic generation is that it simplifies or eliminates the design task, making it possible for programmers or non-designers to create interfaces. Restructuring of designs allow for systematic improvement in usability and aesthetics of interfaces. Prior works have often used rules and heuristics to generate layouts that adhere to specific design guidelines [3, 7, 31, 32, 34], or example-based retargeting by mining existing designs [24, 25]. Such methods often fail to construct objectively better designs that improve performance or usability. Model-based methods, on the other hand, provide principled techniques for generating interfaces that can be measured objectively. Such model-based techniques [5, 7, 34, 39] tackle the design problem by first abstracting the user interface as a set of models, which are then used to steer the generation or restructuring. There have been several prior systems, dating back to the 1980s, such as [4, 20, 28, 33, 42], that apply model-based approaches to interface (re-)design. Our work also situates itself in the area of model-based interface generation.

The above techniques and implementations have largely considered the user population as a whole, and not individuals or groups of users. There has also been some significant work done to address different groups of users. Ability-based design [45] focuses on creating interfaces that take into account different requirements or restrictions, and optimise them towards specific abilities (e.g., [15, 38]). Culture-based design [22] took into account differences among different cultures, and used this as a basis for creating multiple designs of the same interface, each suited to a specific group of users. While all these works have considered a subset of the population, or specific preferences and requirements of groups of users, they generally do not address design on the per-user level, and also do not take into account individual users' past experiences or usage history. In contrast, with SUPPLE, Gajos et al. [12, 14] defined user interface generation as an optimisation problem that took a set of user interactions as input, and used this to generate an optimal solution. By doing so, the system could take into account specific user requirements and abilities, and create individualised interfaces for each user. HIGHLIGHT [27] enabled re-authoring of existing websites based on tracing how the user interacts with the site, and created mobile versions customised to the user tasks and mobile devices. UNIFORM [29] bears a lot of resemblance to our work, as it takes into account a user's history to automatically generated remote control interfaces. UNIFORM differs from our work since it does not restructure existing interfaces, yet focuses on generating consistent user interfaces that provide access to existing functionality on new platforms. Additionally, the user history is limited to one source design, and the work focuses on the engineering aspects of restructuring an interface to match the source, and not on the systematic selection or generation of source designs from an extended history. The general approach adopted by previous works on per-user model-based interface generation has been to model the user as a one-time activity, before generating an optimised interface. Our approach, on the other hand, captures the complete user history, and each time a new interface is encountered, generates a just-in-time redesigned interface based on an updated visual search model for the user. Interfaces can thus evolve over time and usage, and (re-)design becomes a continuous activity.

2.3 Runtime Adaptation of Interfaces

As highlighted above, familiarisation presents an approach to adapt and restructure interfaces at runtime. Such an approach falls under the category of "adaptive user interfaces"—interfaces that can adapt or modify themselves while being used. There have been several works on adaptive UIs previously. Past systems have used different criteria for adapting interfaces, for example, a user's

current physical activity [45], or their current task [13]. In contrast to use scenario or current task, Familiariser adapts an interface layout based on users' history, and exposure to previous interfaces belonging to the same domain. A common criticism and shortcoming of adaptive UIs is that unpredictability and cost of adaptation can negate the benefits provided [16, 26]. Since we focus on adapting newly visited interfaces that the user has not previously used or learnt, it does not suffer from these drawbacks. Thus, by incorporating model-based design generation, and just-in-time interface adaptation, our work attempts to leverage recall and visual learning, and provide users access to interfaces tailored to their expectations and mental models.

3 PRINCIPLES FOR INDIVIDUALISED MODELLING

Searching for features on an interface is a complex cognitive process influenced by several factors related to how people learn visual search. We approach the problem by exploring progressively sophisticated principles inspired by theories of HVS and human memory. These enable us to create a template, consisting of feature positions, for a user.

At the highest level, these templates break down to two groups: (a) interface-wise templates; (b) feature-wise templates. In *interface-wise templates*, an entire interface layout (e.g., webpage) is recognised as one unit. It is assumed that users learn and recall visual layouts in their entirety. On the other hand, feature-wise templates considers high-level features on layouts as individual units. For example, logos, navigation menus, and search-boxes, can be considered as features, and users learn and recall these recurring features across different interfaces. Based on these two cases, we explore four principles.

3.1 Principle I: Frequency

This is the most straightforward *interface-wise* approach to modelling visual search. It is informed by the frequency effect: frequently encountered items are more likely to be recalled than less frequently encountered. More specifically, the number of encounters directly affects both retrieval time and retention probability: more frequently practised items are recalled faster and easier [2]. The interface that has been encountered the most number of times (i.e., most frequently visited), and for the longest durations, is assumed to be the most relevant to the user, and is thus used as the template:

$$S_{interface} = n_{visits} * t_{average}, \tag{1}$$

where $S_{interface}$ is the overall score of an interface, n_{visits} is number of visits, and $t_{average}$ is average duration of visits (in seconds). This results in the selection of one previously seen interface as the template.

3.2 Principle II: Serial Position Curve

While frequency of usage can be a reasonable method to predict the most relevant interface in simple scenarios, as user histories get larger over time, this is susceptible to failure. Usage-based aspects such as first exposure to an interface layout, recency of visits, and intervals between visits, are not considered by the first model. Firstly, not only frequency of rehearsals, but also their recency, affects recall time and probability: recently encountered and practised items are recalled faster and more probably [2]. In addition to recency, there is also an effect of primacy, where items that have been encountered first are better remembered than later items [17]. Together, the primacy and recency effects create a U-shaped curve (a *serial position curve*). It maps items encountered in the past to probability of recall: the first and last items are more likely to be recalled better.

This leads to an *interface-wise* approach that considers the following:

(1) *Frequency (v):* The frequency of visits to a given interface, denoting how often an interface is visited.
(2) *Recency (r):* The recency of visit, denoting when the interface was last visited.
(3) *Primacy (p):* The order of visits to different interfaces, or the sequence in which unique interfaces are encountered.

For each of these, scores are calculated for every interface in the history by ranking them. By aggregating scores for all factors, for each interface layout, we can calculate an overall score for every interface, and hence determine the most relevant interface. Thus, we compute the following scores:

(1) *Frequency Score (S_v):* This is the ratio of the visit count for the given interface to the total visit count of all interfaces in the history.

$$S_v = (n_{interface} * t_{average})/(n_{total} * t_{total}), \tag{2}$$

where S_v is the frequency score, $n_{interface}$ is number of visits to the interface, n_{total} is total number of visits to all interfaces, $t_{average}$ is average visit duration to the interface, and t_{total} is total visit duration to all interfaces.

(2) *Recency Score (S_r):* This takes into account the decaying aspect of memory. The number of other interfaces that are visited since the user last visited the given interface negatively influences the relevance. The score is normalised (most recently visited interface has a score of 1), and this reduces as recency increases.

$$S_r = 1 - r_{interface}/n_{total}, \tag{3}$$

where S_r is the recency score, $r_{interface}$ is recency of a interface, and n_{total} is total number of visits to all interfaces.

(3) *Primacy Score (S_p):* Interfaces that are encountered first tend to be better recalled than later ones. This effect is known as primacy, and the *primacy score* takes this into account, where earlier interfaces have a higher score. The score is normalised (first visited interface has a score of 1), and subsequently reduces for the following interfaces.

$$S_p = 1 - ((p_{interface} - 1)/n_{interfaces}), \tag{4}$$

where S_p is the primacy score, $p_{interface}$ is visit order number for the interface, and $n_{interfaces}$ is number of unique interfaces visited.

By aggregating these scores, we derive the overall score for an interface:

$$S_{interface} = \alpha * S_v + \beta * S_r + \gamma * S_p, \tag{5}$$

where $S_{interface}$ is the overall score for the interface, α, β, γ are weights for the three factors, and $\alpha + \beta + \gamma = 1$.

In our implementation, we make the assumption that frequency and recency contribute equally towards familiarity, and suggest that primacy has a lower effect. Taking this into account, we set the weights as $\alpha = \beta = 0.4$, $\gamma = 0.2$. Intuitively, this weighting makes sense in our system. However, these weights could be further fine-tuned by empirical testing, or can be made adjustable by end-users, if desired.

The interface with the highest overall score ($max\ S_{interface}$) is considered to be the relevant interface for the user, and is selected as the template.

Figure 3 presents a sample scenario, where a user visits four unique interfaces multiple times. As the user visits different interfaces, Figure 3(a) illustrates evolution in recorded usage metrics and scores, along with interface selection results as per principles I and II.

3.3 Principle III: Visual Statistical Learning

This definition takes into account the statistical frequency with which different design features occurred in history H. This hypothesis is based on theories of visual statistical learning [9, 43], according to which visual search strategies are sensitive to the statistical structure of the visual environment.

A *feature* is defined as a high-level semantic element, and can be composed of multiple low-level elements. For example, a website's logo is a feature, and could be composed of an image and a link element. By aggregating the commonly found positions of each feature among the visited interfaces in a user's history, we can determine the *most probable* position for each feature, and use this to create a template. Unlike in the previous principles, the resultant template here is not a single interface from the history, but a *feature mesh* consisting of features from multiple interfaces in the user's history.

First, each page is assigned a score based on the serial-position curve (from Principle III):

$$S_{interface} = \alpha * S_v + \beta * S_r + \gamma * S_p. \tag{5}$$

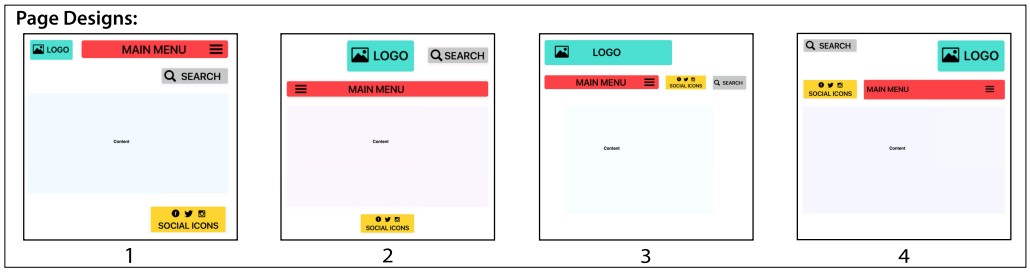
To generate the feature mesh, and template, we first generate spatial probability distribution maps for each encountered feature. In these maps, for interfaces found in the history, for every pixel occupied by a particular feature, the spatial probability score (S_{ij}) is incremented. Features are weighed according to overall scores ($S_{interface}$) of each interface (from III). Thus, a feature appearing on a more relevant interface has a higher influence on the probability distribution. The following pseudocode outlines the method for computing scores for each pixel on the spatial probability distribution map:

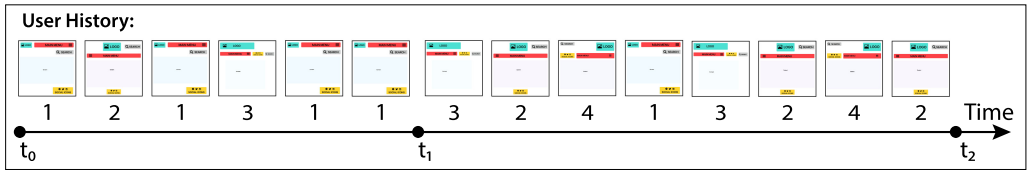
```
for Feature F in AllFeatures do                          // Iterate over features
    for Interface I in InterfacesInHistory do            // Iterate over interfaces
        if F ∈ I[Features] then                  // Check if feature F is in interface I
            for i in F_I[yPos] : F_I[yPos + height] do
                for j in F_I[xPos] : F_I[xPos + width] do
                    S[i][j] := s[i][j] + S_interface;    // Increment score for pixel ij
                end
            end
    end
end
```

Once the feature maps are constructed, we find the positions with the highest score:

$$\max_{0 \le i \le height} \max_{0 \le j \le width} S_{ij}.$$

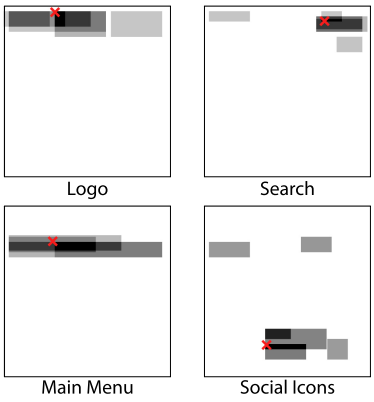
For each feature, this position (ij) is selected to create a template consisting of all detected features. Figure 3(b) illustrates the computed probability distribution maps and selected positions for different features, for the given scenario.



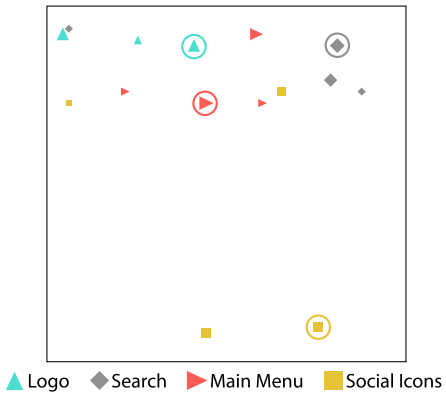


Timestamp	Page	Frequency (v)	Recency (r)	Primacy (p)	Frequency Score (S_v)	Recency Score (S_r)	Primacy Score (S_p)	Familiarity Score (F_{page})	
t_1	**1**	**3**	0	1	0.667	1.000	1.000	**0.867**	← Selected by I and II
	2	1	4	2	0.167	0.333	0.667	0.333	
	3	1	2	3	0.167	0.666	0.333	0.400	
	4	0	-	-	-	-	-	0	
t_2	**1**	**5**	4	1	0.357	0.714	1.000	**0.629**	← Selected by I
	2	4	0	2	0.286	1.000	0.750	**0.664**	← Selected by II
	3	3	3	3	0.214	0.786	0.500	0.500	
	4	2	2	4	0.143	0.857	0.250	0.479	

(a) Usage Metrics and Scores



(b) Probability Distribution Maps (for III)



(c) Activations (for IV)

Fig. 3. Scenario with four different interface layouts. The user history shows a timeline of interface visits (for simplicity, uniform visit durations are assumed across interfaces). (a) Evolution of metrics and scores at two timestamps (t_1, t_2), and interface selection outcomes. (b) Computed probability distribution maps for key features at t_2. Red × symbols indicate the selected position for each feature. (c) Activations for features at t_2. Circled activations indicate the predicted positions for each feature.

3.4 Principle IV: Visual Sampling Based on a Generative Cognitive Model

This definition uses a generative approach, where a model of visual search is used to generate gaze fixations as the simulated user is searching targets on an interface layout. The simulation integrates models of eye movements, visual short-term memory, and associative long-term memory, and proposes that visual search is the interaction of these three resources [21]. Given a user history with interfaces, including locations of elements on the layouts, the model generates eye movement patterns as it simulates human-like visual search on the interfaces. Using the model of eye movements [37], the simulation encodes elements on the interface until it has found the target. For each element, the encoding time is

$$T_e = K \cdot [-log(f)] \cdot e^{k \cdot \epsilon}, \tag{6}$$

where f is the frequency of the object, either supplied externally to the model or assumed to be uniform, ϵ is the distance of the target from current eye fixation, and K and k are scaling constants, adapted from the literature [37]. T_e increases exponentially as the target is further from the fixation, but the visual system may compensate this by initiating a fast eye movement to gaze closer to the target, with movement time of

$$T_s = t_{prep} + t_{exec} + D \cdot t_{sacc}, \tag{7}$$

where t_{prep}, t_{exec}, and t_{sacc} are constants from literature [37] and D is the movement amplitude.

After the eye movement, the target needs to be encoded (6). However, if the encoding time is less than t_{prep}, then the target is encoded without the eyes moving. This creates an eye movement model where the eyes move only when the target is too far to encode from the current gaze point.

In addition to visual search, the model simulates learning of layouts by using an activation-based associative memory model [1]; its location is stored in the memory storage. An activation strength of an association can be calculated based on the number of times the target has been found:

$$B_i = \ln\left(\sum_{j=1}^{n} t_j^{-d}\right), \tag{8}$$

where t_j is the time since the jth time of finding the target i, and d is a decay parameter, set from literature [21]. As the model learns the layout, it can use the associative memory to recall the position of the target without having to visually search for it. The model is able to recall the position of the target, if $B_i > 0$, with noise added from normal distribution [21]. Recall time is

$$T_i = Fe^{-fB_i}, \tag{9}$$

where F and f are scaling constants, set based on literature [21].

Longer history with a layout results in higher associative activations, and faster, more expert-like performance in finding targets. From the activations, the model can predict, where the user will gaze, given a layout and user history, to search for a particular feature. The most prominent activation points are selected as feature positions, to create a template for the user. Figure 3(c) illustrates activations, and predicted positions, for the given scenario.

4 FAMILIARISATION: RESTRUCTURING LAYOUTS TO REDUCE VISUAL SEARCH

By creating individualised models of visual search, we can enable restructuring of layouts, and realise our goal of reducing the effort for finding features on new, previous unseen, interfaces. We present *Familiarisation* as one such instantiation, where these principles are applied to restructure and *familiarise* new graphical layouts. Familiarisation takes advantage of application and operating system capabilities to record user history, and render content dynamically, in their original or

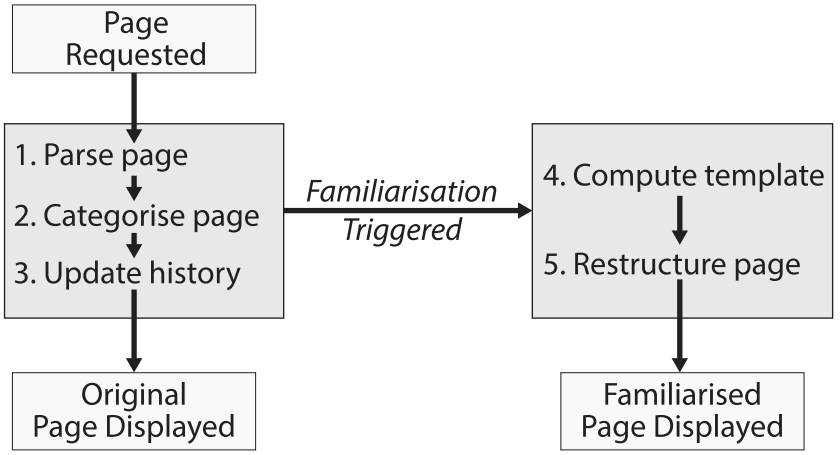


Fig. 4. Familiariser Pipeline: When a page is requested, it is parsed and categorised, and the history is updated for visited pages. Once familiarisation is triggered, the template design is computed using an HVS principle, and restructuring is instantiated via layout optimisation prior to rendering on a web browser.

modified form. This enables us to change an interface layout before it is displayed to the user, such that it appears familiar, making features more easily findable.

There are three essential components required for a system to implement familiarisation:

(1) **Logging:** The system must be capable of capturing and logging a user's interaction history. It should record information about each interface visited, and the structure of the features on the layout. Additional metadata of the visit also need to be stored, such as visit duration, number of previous visits, and time since last visit. This information is required for modelling and creating a template for the user.
(2) **Template Modelling**: The system uses the history logs to create a template design for the user. It needs to iterate over previously visited interfaces, and cumulate the usage information to select or generate an individualised template for the user. The template is chosen using the principles described in the previous section.
(3) **Layout Restructuring**: When the user visits a new interface, the system should restructure its layout before it is rendered. For restructuring, it applies the individualised template to reposition features found on the new interface. Additionally, the system needs to ensure the restructured layout is valid by placing unmatched elements, and resolving any overlaps of elements.

Any system that is capable of implementing these three components can support familiarisation. As one such case, we implemented *Familiariser*, a browser-styled application to familiarise website layouts. The following sections describe the pipeline and implementation of Familiariser.

5 FAMILIARISER: BROWSER-SIDE WEBPAGE RESTRUCTURING

We implemented the concept of familiarisation in *Familiariser*, a browser-styled application that allows users to visit webpages and enables access to familiarised versions, adapted towards each user. As illustrated in Figure 4, when a user visits a webpage, the following pipeline is executed:

(1) **Parse Page:** The page source (HTML) is parsed by the system. Key elements, representing high-level features, are detected and labelled.

(2) **Categorise Page:** The page is classified into a general website category (e.g., shopping, banking, travel).
(3) **Update History:** If the visited page does not exist in the history, it is added; if it was previously visited, the corresponding entry in the history is updated for the page. Additionally, familiarity scores, described in the previous section, are updated for every page in the user history.

While familiarisation is disabled, the pipeline ends at this point, and the original page is displayed as is. Once familiarisation is triggered, the following steps ((3) and (4)) are executed to familiarise the page.

(4) **Compute Template Design:** The history is filtered for pages belonging to the same category. Based on this, we compute the template design, to be used as the basis for restructuring the new page.
(5) **Restructure Page:** The new page is restructured using positional values of matching features from the template, and by repositioning the corresponding elements on the visited page. Overlaps may occur in the restructured page. Familiariser resolves any such overlaps while attempting to best maintain relative alignments of elements on the page. This ensures a valid layout, without obscuring or omitting any contents. The familiarised page is finally displayed to the user.

Figures 5 and 6 illustrate two examples of results obtained by Familiariser, given a user history and original (unfamiliar) page, for each of the four presented principles.

In the remainder of this section, we describe the various components of the system in greater detail.

5.1 Page Parsing

A *feature* can be defined as a task-level element [44]. They are actionable elements, with a well-defined purpose, such as the logo, search bar, buttons, or icons to log in and access a user account, or the shopping cart on e-commerce sites, and so forth. Features can be composed of several low-level HTML DOM elements. For familiarisation, it is necessary to first detect these features on a given page. Prior works have explored element detection when underlying sources were not available. Prefab [10] presented a pixel-based approach to reverse-engineering the GUI. Sikuli [46] allowed users to take screenshots of widgets and elements, and use these for search and automation of visual interfaces. For webpages, however, the underlying source files are openly accessible. Previously, Webzeitgeist [24] used CSS selectors to detect common features. Familiariser parses the underlying DOM tree of a page, and analyses DOM tags, identifiers, and linked CSS classes, of elements to automatically detect features on the page. We use partial string matching to detect commonly used names, and map them to corresponding features. A feature can either be a leaf node in the DOM tree, or a compound element consisting of several smaller elements. The detection of features relies on appropriate naming and tagging of underlying HTML elements, and this can be a limiting factor in finding all matching features across different pages. Feature recognition could be improved by employing other computational methods, such as image recognition or machine learning, but this is out of the scope of this work.

5.2 Page Categorisation

A *category* defines a group of pages that semantically or functionally belong together, and are used in similar ways or for similar tasks. Since features on different categories of websites tend to have different properties or aesthetics, it is important to segregate them. Therefore, each page is assigned a textual category label once it has been parsed. Familiariser uses HTML tag annotations

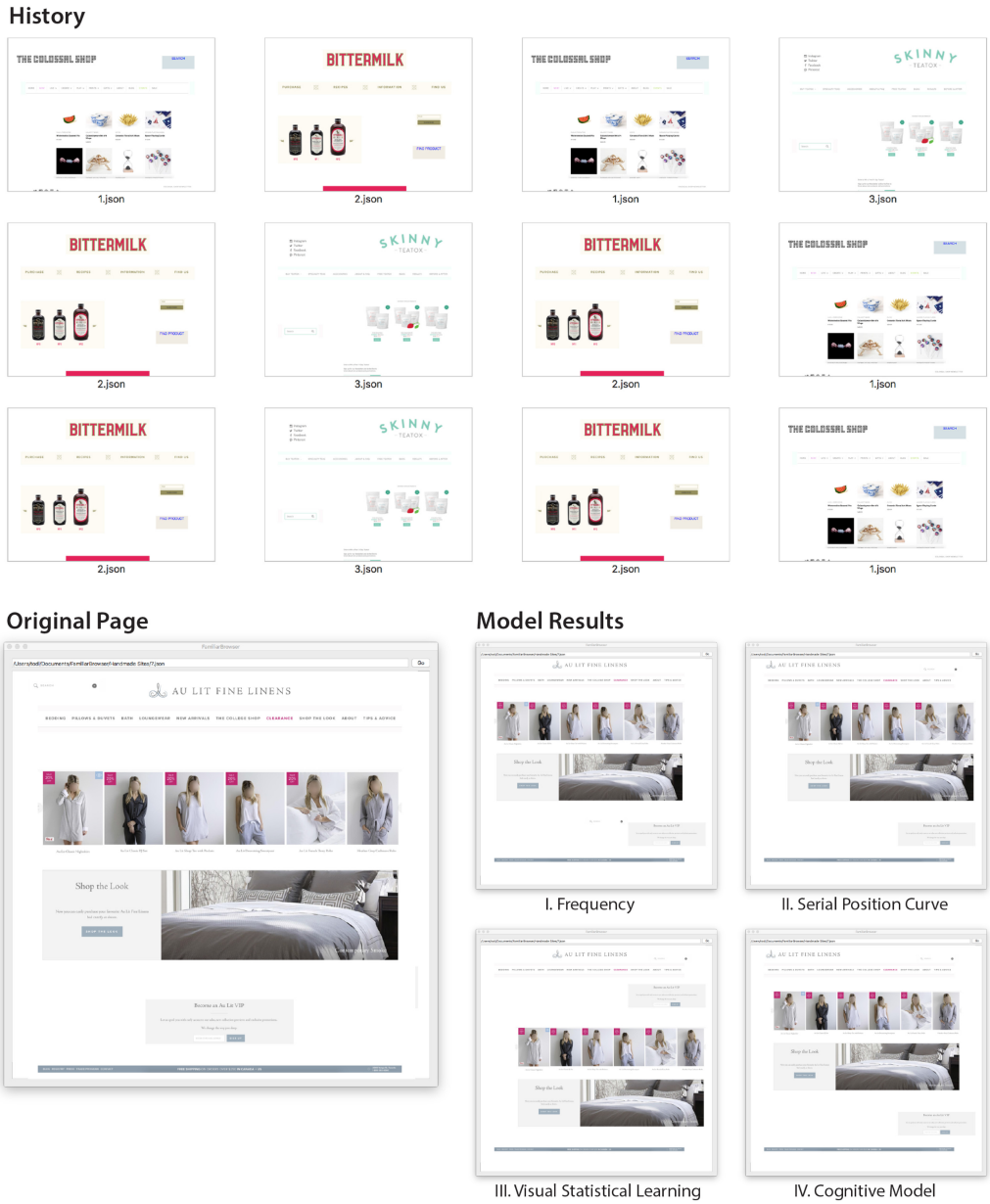


Fig. 5. A visual comparison of the results after familiarising a page using the four presented principles. For a given user history, when a new page is visited, each of the models may produce varying results.

to categorise visited pages into general categories. Automatic categorisation can be improved by using other strategies such as topic modelling [6], or by matching features found across pages.

5.3 Usage History Updates

For each user, page visit history is recorded, and usage-based metrics are updated each time new pages are visited. For each page, a unique entry in the history is maintained, and includes

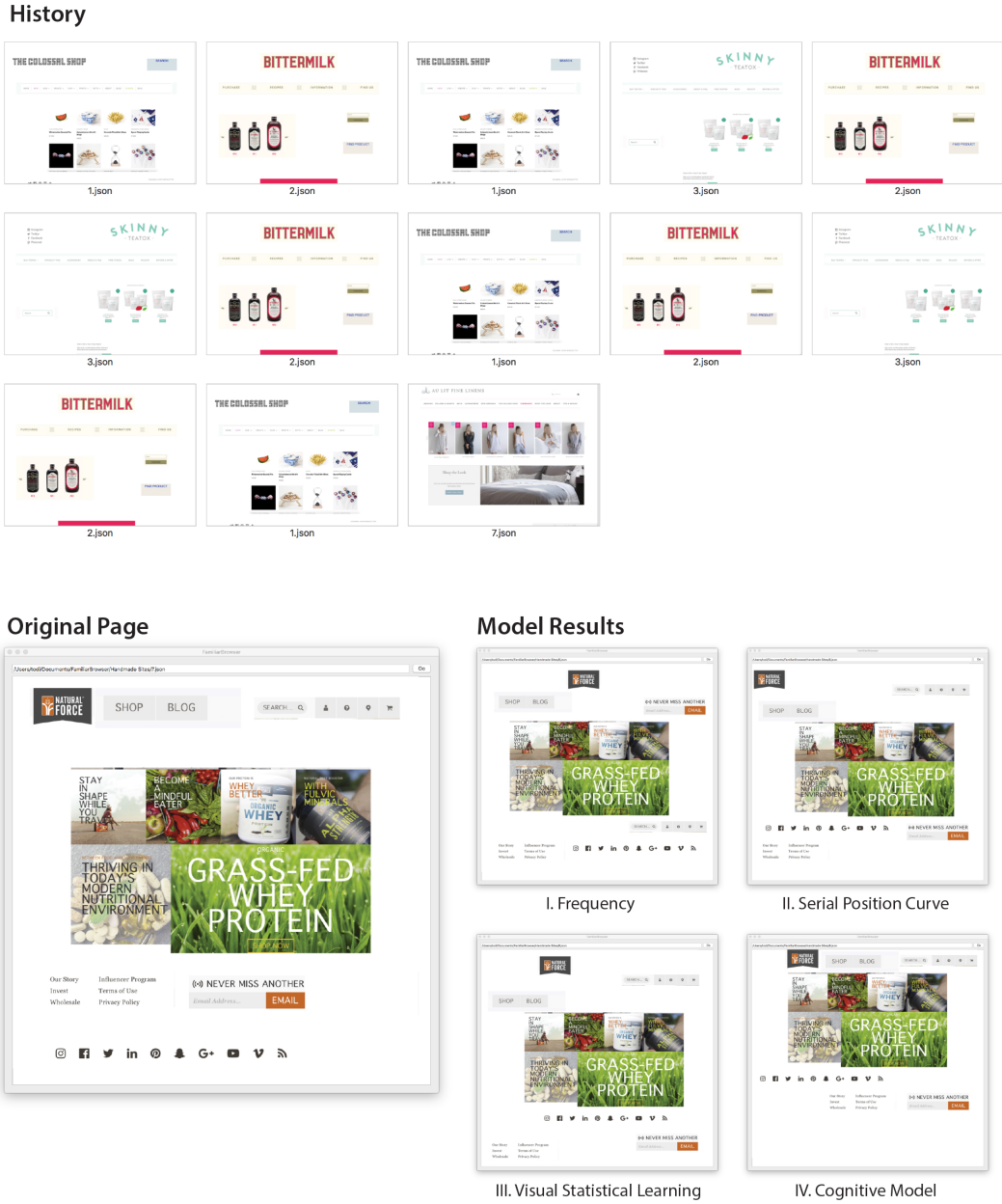


Fig. 6. A second example of results from restructuring a new page. In this case, the user has visited additional pages, indicated in the updated history. The template is recomputed, and used to create familiarised results.

information related to time spent on the page (duration), frequency of visits, recency, and order of visits. When an existing page in history is re-visited, the frequency (f_{page}) is incremented, and the recency of that page is reset such that it is the "most recent" page (i.e., $r_{page} = 0$). For every other page in the history, the recency (r_{page}) is incremented, thus decreasing the recency score (S_r).

5.4 Familiar Design Computation

All pages in the history are considered to compute the familiar layout design. For web browsing specifically, different categories of websites have distinctly different features. To avoid mismatch in domain, website categorisation can be performed as an intermediate step, and only pages within the same category as the currently visited page are considered. For example, if the user intends to visit a shopping website, only pages from the shopping domain are considered while computing the base layout design.

Principles I and II consider a single page as the "familiar design." Familiarity scores are calculated accordingly, and the highest-scoring page is selected. Matching features are extracted from the familiar design, and their xy positions are used for the base layout design. Principles III and IV compute familiar designs based on all pages in the user history. For principle III, probability distribution maps are created for features detected on all visited pages. Using these, the most likely position for each feature is estimated, resulting in a base design where all features are located appropriately. For principle IV, activation points for each feature are computed. The point with the highest activation, for a feature, is predicted to be the most familiar position, and hence used for the base design.

5.5 Target Page Restructuring

The base design that is computed is used to restructure the newly visited page. Features on the target page are matched to those on the base design, and repositioned accordingly. In [25], retargeting occasionally resulted in truncated or cropped content due to size mismatch. In our approach, while repositioning, dimensions of the original elements are maintained so as to avoid truncation of contents. This can, however, result in some overlapping or occlusion of elements.

Overlap Resolution. Overlaps are resolved by setting up a series of overlap-redressal rules. Examples of rules used in Familiariser are as follows:

(1) Left-alignment or top-alignment of two (or more) elements should not be violated. If this is violated, a penalty is applied.
(2) Movement of any element from its preferred (horizontal) location entails a penalty.
(3) The vertical or horizontal sequence of any pair of elements should be honoured.
(4) The canvas width should not be changed. The height may be changed if needed.

The rules are implemented using any standard integer-linear programming solver, resulting in a valid layout without overlaps.

5.6 Triggering Familiarisation

During the initial stages of browsing, the system updates its model, and displays the page in its original form. As users visit different pages, they gradually learn visual layouts of these, thus increasing chances of recall during future visits. Once a user is fluent with a small set of pages, familiarisation is triggered, and newly visited pages are adapted to match the computed template design. However, we need to determine the ideal moment to trigger familiarisation, to make it effective. There is a tradeoff between familiarising too early and too late. If triggered too early, users may not have learnt visited pages sufficiently, thus future pages would not benefit from adaptation. On the other hand, if familiarisation is delayed, then users might have already been exposed to a large number of diverse designs, thus making recall harder. For the purpose of our study, we determined (by trial-and-error) that enabling familiarisation after 25 page visits was favourable, given that the number of unique pages was less than or equal to 5. For future systems,

familiarity scores can be used to empirically determine when to trigger familiarisation, or this could be customisable per user.

With automatic (always-on) familiarisation, pages are automatically adapted when they are requested. Familiariser also supports manual (on-demand) familiarisation, where users can explicitly request for a familiarised version of a page. Here, the original page is displayed by default, and a familiarised version is rendered only when demanded. This is similar to how web services such as translation and reader-friendly modes enable users to access adapted versions of a page. Manual familiarisation obviates the need for determining a fixed point at which adaptations are triggered.

6 ARCHITECTURE AND IMPLEMENTATION

The front-end of Familiariser is implemented in Swift, on MacOS 10.13, as a standalone browser-styled application. The back-end of our the system consists of three main components: logging user history, generating a familiar design, and restructuring the page.

6.1 Logging User History

The system maintains a persistent history file for the user. Every time a new page is visited, custom JavaScript code parses the page source, and converts the page into a flattened JSON file. The JSON file is parsed to construct a webpage object with the following internal model:

(1) *URL (String):* Unique address identifying the website.
(2) *Category (String):* The general category of websites to which the page belongs; for example, shopping, banking, travel, social networks, and so on.
(3) *Page Elements (Array):* Content elements appearing on the webpage. Each element contains absolute (xy) positions as they appear on the browser, feature name, and other element-specific properties, such as tags and CSS styles, required to recreate the page.
(4) *Primacy (Integer):* The visit order number indicating the primacy of the page.
(5) *Frequency (Integer):* Number of times the page has been visited.
(6) *Recency (Integer):* A number indicating how many other pages have been visited since the last visit.

When the user completes a page visit, either by navigating to a different page or by quitting the application, a new page record is added sequentially to the user history. This record contains an identifier for the webpage, a pointer to the internal webpage object, a timestamp indicating when the page was visited, and time duration of the visit. This sequential history contains all information required to model a user's familiarity.

It should be noted that when the system is first used, the history is empty, and thus contains no information about pages the user might have seen before using the system. While the system can learn quickly from the user's consequential visits, it could be advantageous to include some information about commonly visited pages that the user might already be familiar with. This can be made possible by using a data-driven approach, where popular pages have a preset familiarity score at the initial stage.

6.2 Generating a Template

When familiarisation is triggered, the system filters the history for all pages belonging to the same website category, and creates a familiar template unique to each category.

For *frequency-based familiarisation (I)*, the system finds the webpage with highest frequency in the user history, and selects it as the familiar design.

For *serial-position curve (II)*, it calculates the familiarity scores for each webpage in the history, and selects the page with the highest score.

For *visual statistical learning (III)*, the system initialises an empty 2×2 array, with pixel dimensions of the largest webpage in the history. It iterates over all pages in the history, to create probability distribution maps by assigning appropriate pixels a weight, when a feature is found. Next, for each feature, it iterates over the respective map, and finds the pixel with the highest value. The final familiar design consists of (x, y) points with highest probabilities for all found features.

The *cognitive learning model (IV)* is implemented using Common Lisp. Each time familiarisation is required, a CSV file containing relevant history information, including access timestamp, duration, and linked JSON file name, is generated. The CSV file and required JSON files for all webpages are passed as input to the learning model. As output, the model generates a new CSV file containing activation points (x, y) for all detected features, along with confidence values, and returns this to Familiariser. Familiariser uses this to select the desirable positions for features, to generate the familiar design.

6.3 Restructuring the Page

Given the familiar design, Familiariser automatically restructures a newly visited page by repositioning matching features at locations found on the familiar design. Any unmatched features are then positioned at their original location. Next, overlaps in layout elements are resolved by passing a file containing element positions to a Java application. The application applies overlap-redressal rules to each element. We use an IBM CPLEX linear programming solver to optimally resolve any detected overlaps. The solver returns a corresponding file with resultant element positions. This is used by Familiariser to generate the final valid layout, which is then displayed to the user in the browser-based application.

7 COMPUTATIONAL AND COGNITIVE COSTS

Each of the four principles presented in Section 3 has fundamentally different computational properties. When implementing them in a system, these differences also have an impact on the computation time required to choose or create a template design.

Given a user's history, consisting of N unique interfaces, and M total visits, we can summarise the performance properties as follows:

(1) **Frequency:** To implement the frequency-based principle, a system records the numbers of visits to each interface. When selecting a template for the user, it simply needs to iterate over the visited interfaces, to find the interface with the highest frequency. Therefore, N iterations are required to select a template design, and the computation time is negligible.
(2) **Serial-Position Curve:** To implement the serial-position curve, a system captures usage information for each interface visited. When an interface is visited, it updates metrics such as primacy, frequency of visits, and recency, and computes an overall score for each of the N interfaces in the history. Therefore, at every visit, N iterations are required to update the metrics. Further, to select a template, it iterates over the N interfaces to find the highest overall score, and chooses it as the template. Like in the first principle, this is computationally simple, and the computation is nearly instantaneous.
(3) **Visual Statistical Learning:** The visual statistical learning principle uses probability distribution maps to select positions for each feature. To generate these, it starts with an empty 2×2 array, whose dimensions are the same as the canvas size (in pixels) of an interface layout. For each feature, it iterates over the N interfaces, and checks if the feature is found. If present, it increments the value of the corresponding positions, pixel-by-pixel, in the feature matrix. Once all probability distribution maps are generated, it iterates over each of them to find the most likely position for the feature, and thus creates the template.

Since the system needs to iterate over several two-dimensional arrays, pixel-by-pixel, this is computationally more expensive. The exact computational time is dependent on several factors such as the number N of interfaces, the number of features on each interface, and the size of the canvas. This can vary greatly from case-to-case. A complete formalisation of the computation time is out of scope of this article; however, this can be reduced by considering grid structures of layouts, and grouping pixels. Further, since computation of the template can happen every time an interface is visited, the template can be pre-computed, and immediately applied when a new interface is visited.

(4) **Generative Cognitive Model:** To implement the visual sampling-based cognitive model, the system records sequential history of M visits to the N interfaces, along with corresponding timestamps. Given the position of features, the model generates gaze fixations for the duration of visits, and updates this for each of the M visits. Therefore, it iterates M times to generate all activation points. The template is created by selecting the most-prominent activation point for each feature found on the interfaces. The computation time is, therefore, influenced by the total number of interface visits (M), and the duration of visits.

The principles can also imply different mental requirements for the user, because they involve different cognitive mechanisms in their approach to what familarity means. For the first two principles, the familiarised layout is an actual, previously seen layout, whereas for the two latter principles, a completely new layout is generated. This means that in the previous case, the mental mechanism that supposedly produces efficiency due to familiarity is *recognition.* Humans are very quick at recognising previously encountered visual stimuli, even when a stimulus is encountered only once and very briefly [18]. For the last two principles, there is no specific layout to be quickly recognised. However, because the elements are close to their expected and familiar locations, the user should have no trouble finding them. Whereas the first case (principles I and II) was about retrieving a full visual representation from the memory, the second case (principles III and IV) is about retrieving the location of the specific type of an element from memory and using it to help construct a new mental representation. It can therefore be hypothesised that the former case might be slightly faster for the users, because it does not require creating a new mental representation, for the same reason it is also less flexible, and requires recognition of the original stimulus.

8 LABORATORY EVALUATION

The goal of our evaluation is to verify the concept of familiarisation, and the presented Familiariser system. We seek to answer the question: *Does familiarisation improve performance in web browsing for end-users?* To this end, we conducted a controlled user experiment, and report on quantitative results. We compared original (unmodified) designs against familiarised designs in a study where users were asked to point-and-click on different features on the displayed page. For the task of web browsing and selection, the frequency and serial position curve principles have some shortcomings. The frequency principle does not take into account evolution in user's browsing behaviour, while the serial position curve does not address similarities in feature distributions across common webpages. Given the short duration of page visits during this experiment, the generative cognitive model is provided with rapid interaction bursts, that last only a few seconds. Further, the number of visits to each page is quite limited. Consequently, generated activation points are located near the most frequently occurring positions of features. This, in nature, resembles results from the visual statistical learning (principle III). The similarity in results from the two models, under typical study conditions, was verified by visually eye-balling some resulting layouts with both

principles. As visual statistical learning is computationally less expensive, we used this principle for familiarisation during the study.

As dependent variables, we analysed visual search time, approximated by pointing time, and number of eye-gaze fixations per target feature. Comparing these two cases enables us to evaluate the potential effects of familiarisation during real-world usage. For the test case, we chose the domain of shopping websites, a frequently used category of webpages that are also plenty in number. These websites typically contain similar features yet vary vastly in their layouts and presentation of these features. This makes them a good test candidate for Familiariser, and provides a realistic use scenario for the study.

8.1 Study Tasks

For the study, participants were given the task of selecting (clicking on) a feature element on the displayed webpage, in a browser-styled application. Webpages were selected randomly, from a dataset of 30 shopping sites. The target feature was also selected randomly from the page, and included commonly occurring elements, such as logos, navigation menus, search boxes, among others. Participants were requested to perform tasks quickly yet as accurately as possible, and avoid unnecessary pauses. As tasks were performed, the software logged mouse movements, click events, and eye-gaze information, including eye position (averaged) and fixations.

8.2 Apparatus

A 13′′ MacBook Pro with Retina Display (2,560-by-1,600 pixels), running MacOS 10.13, was used for the study. The browser-based Familiariser software was displayed as a full-screen application. We selected 30 shopping websites to create the dataset, excluding commonly used shopping pages to avoid bias. Webpages used in the dataset were rendered offline to avoid delays. Tasks were completed using the in-built trackpad. All cursor motions and events were logged by the study software. For eye-tracking, an EyeTribe Tracker[1] was used, along with a custom Python program to log gaze positions and fixations.

8.3 Participants

We recruited 16 participants, aged 21 to 36 (mean 29), with no visual impairments. All participants reported frequent web usage (daily), and also reported exposure to shopping websites. Participants completed tasks with both original (unmodified) and test (familiarised) pages, in a within-subject study design.

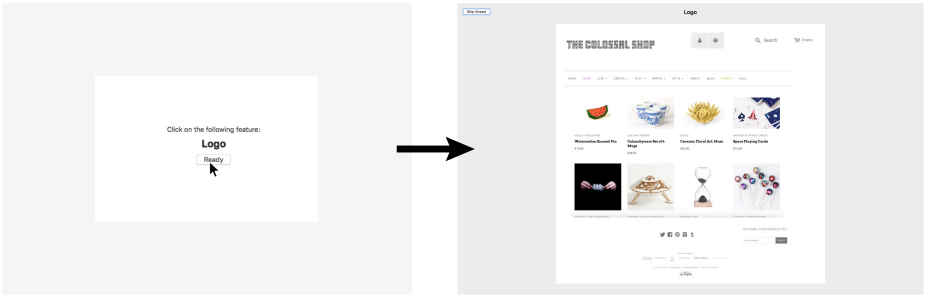
8.4 Method

During the experiment, participants were exposed to a set of different websites. Before a webpage was displayed, they were presented with the task of selecting (clicking on) a particular feature element (e.g., logo, main menu, search box) on the page. The target feature element was selected at random from the page to be displayed. The timed trial started once the participant confirmed they were ready, by clicking on a confirmation button. This button was consistently placed at the centre of the screen, ensuring a constant starting point for the cursor during all trials. Figure 7 shows screenshots of this setup.

The experiment was divided into two phases: *Learning Phase* and *Test Phase*, as described below.

(1) **Learning Phase:** For each participant, the system selected 5 webpages, at random, from the dataset of 30 pages, for the learning phase. The learning dataset thus varied for each

[1] www.theeyetribe.com.



Task Stimulus Page Displayed

Fig. 7. Screenshots from one trial of the study. The user is first shown a stimulus, with the task description. The task is started when the user clicks on the "Ready" button, thus positioning the cursor on the screen centre. A page is then displayed; the task is completed when the user successfully clicks on the target.

Table 1. Summary of Results for Average Visual Search Time and Fixation Count per Target Feature

	Visual Search Time	Fixation Count
Original	2.8 seconds	3.4
Familiarised	2.5 seconds	2.6

participant. During each trial, a page from this subset was selected at random, and the participant was asked to point and click at a randomly chosen feature on the page. A total of 25 trials were performed during this phase, and this was used to construct the "usage history" for the participant. We chose 5 pages for the learning set, as this allowed participants to get familiar with them within a short duration of time.

(2) **Test Phase:** Once the participant had experienced this subset of pages in the training phase, the experiment moved on to the test phase. The five pages used in the learning phase were stored in the user history, and excluded from the test dataset. Similar to the learning phase, participants again browsed to randomly selected pages, and were asked to select a target feature on the page. During the test trials, either the original (unmodified) page or a familiarised version was presented to the user, chosen at random. Participants performed a total of 100 timed trials during the test phase.

The average duration of the study was approximately 30 minutes for each participant.

8.5 Results

We created a linear mixed model with search time as dependent variable, page type (Original vs. Familiarised) as fixed independent variable, and participant id and page URL as random variables. Table 1 summarises the key results. Grand mean search times were Original = 2.8 seconds and Familiarised = 2.5 seconds. The difference was statistically significant, $t(663) = 5.3, p < 0.001$, with model $F(1, 663) = 28.5, p < 0.001$. In standardised units, the benefit of familiarised layout was $\beta = 0.35$.

We compared similarly the number of fixations per target. Grand mean fixation counts were Original = 3.4 and Familiarised = 2.6. The difference was statistically significant, $t(596) = 4.7, p < 0.001$, with model $F(1, 596) = 22.3, p < 0.001$. In standardised units, the benefit of familiarised layout was $\beta = 0.35$.

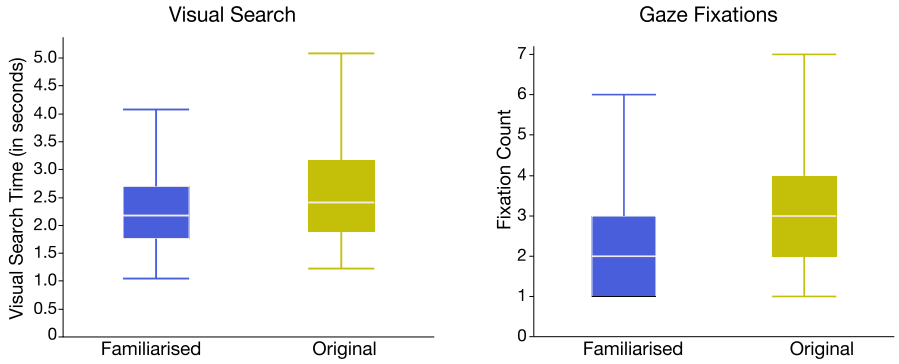


Fig. 8. Boxplot graphs of results for visual search time, and eye-gaze fixations. On average, familiarised layouts have lower search times and fewer fixations.

The above results (Figure 8) indicate that familiarisation improved user performance by reducing both visual search time and number of gaze fixations. Our study evaluates familiarisation using only the visual statistical learning principle.

9 DISCUSSION

In this article, we formalised principles for individualised modelling of visual search on graphical user interfaces. This work situates itself in the field of automatically generated user interfaces that adapt to individual users. The explored principles have their foundation in cognitive scientific theories of how the human visual system learns an interface and recalls it over a longer period of time. This is key to predicting how users visually search new, previously unseen, interfaces for known features. By modelling this, we can choose or create user-specific templates that predict the most expected locations of features, and use this to improve interface layouts. While optimisation techniques and approaches have long sought to improve interface designs universally, for entire populations, our work creates individualised designs improved for each user. Previous approaches to adaptive UIs have also had the drawbacks that they are often perceived as distracting, or conflict with user's expectations, thus adding a cost of adaptation. Our modelling and restructuring approach also avoids this pitfall by avoiding the adaptation of previously learnt layouts.

As one use case, we present an approach for *familiarisation*, where the user history is sampled to create a template, and applied to restructure new graphical layouts. The interface can now be restructured to appear closer to each user's expectations. This concept was implemented in *Familiariser*, an end-user system that captures users' history, and restructures newly visited pages based on automatically generated template designs. Results from the empirical study provide evidence for the approach in realistic conditions. For web browsing tasks, familiarisation significantly improved visual search time by over 10%, and reduces the number of fixations by over 23%, while searching for features on several pages.

The cost of adaptation is often a concern for adaptive user interfaces. Our work circumvents this as familiarisation restructures only new and unfamiliar designs, instead of continuously adapting or modifying frequently used layouts. One could criticise our approach for compromising brand identity, or undermining the designer, by modifying designs. However, we argue that usability of an interface supersedes these aspects for the end-user. Moreover, one could argue that improving the searchability of brand-related elements with familiarisation can improve brand perception. Additionally, Familiariser addresses this by allowing users to optionally view either original or familiarised versions of designs. Commercial browsers have also used such techniques to improve

usability of webpages. For instance, "reader-friendly mode" on browsers allows users to switch between the original page and a version enhanced for reading.

There are limitations for applying our principles universally, as it requires (1) logging of a user's history of seen layouts, (2) detailed information and representation of interfaces, (3) just-in-time computations of a template design, and (4) instantiation of layout restructuring in runtime, prior to rendering it on a display. By exploring a range of principles, varying in computational complexities, we provide alternatives that enable systems to circumvent some of these limitations. The basic frequency-based approach (principle I) is straightforward to implement, and requires minimal user history information. Serial-position curve (II) requires some additional usage information, and requires calculations of various scores to select a page as the familiar design. The feature-based principles (III and IV) require detailed information about the interface layout, and are computationally more expensive, but offer more accurate representations for a user. Given these considerations, dependent on the usage scenario and intended application, one of the four principles can be selected to achieve familiarisation. While our laboratory evaluation illustrates the benefits of familiarising webpages using visual statistical learning, a full comparison of all four principles requires a more extensive study. More specifically, to fully assess the impacts of the different models, a longitudinal study would be necessary, as extensive amounts of data would be required. Additionally, real-orld usage can vary from user to user, and this is difficult to capture during a controlled experiment. Thus, a field ("in the wild") study would be an ideal format for such a comparative experiment.

While Familiariser is one instantiation focusing on repositioning of features, future efforts can apply our work to address other interface aspects, such as colours, fonts, and other visuo-perceptual properties. Such properties are often position-agnostic, and thus for these interface-wise templates (I or II) might be preferred. Apart from addressing additional interface properties, usability and experience with user interfaces can be further improved. Future systems can explore a dual-optimisation strategy, where the presented model can be combined with other predictive models of human perception. Additionally, more interfaces can be covered by applying the principles to a variety of mediums and interfaces (digital, physical, hybrid). For instance, it could be possible to restructure physical interfaces, adapting them to resemble previously encountered digital or physical interfaces. Finally, we can also apply the principles to achieve goals such as diversification or exploration of interfaces. In this case, instead of matching an interface to the template, explicit effort could be made to position features such that they do not align with the template, thus drawing more attention to them.

More information and material related to this article can be found at http://www.kashyaptodi.com/familiarisation.

ACKNOWLEDGMENTS

Among others, we thank Niraj Damaya for code related to overlap resolution, Markku Laine for his useful comments, and study participants for their time and involvement.

REFERENCES

[1] John R. Anderson, D. Bothell, C. Lebiere, and M. Matessa. 1998. An integrated theory of list memory. *Journal of Memory and Language* 38, 4 (1998), 341–380. DOI: https://doi.org/10.1006/jmla.1997.2553

[2] John R. Anderson, Jon M. Fincham, and Scott Douglass. 1999. Practice and retention: A unifying analysis. *Journal of Experimental Psychology-Learning Memory and Cognition* 25, 5 (1999), 1120–1136.

[3] Yigal Arens, Lawrence Miller, Stuart C. Shapiro, and Norman K. Sondheimer. 1988. Automatic construction of user-interface displays. In *Proceedings of the 7th AAAI National Conference on Artificial Intelligence (AAAI'88)*. AAAI Press, 808–813. http://dl.acm.org/citation.cfm?id=2887965.2888108

[4] Gilles Bailly, Antti Oulasvirta, Timo Kötzing, and Sabrina Hoppe. 2013. MenuOptimizer: Interactive optimization of menu systems. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST'13)*. ACM, New York, NY, 331–342. DOI: https://doi.org/10.1145/2501988.2502024

[5] C. M. Beshers and S. Feiner. 1989. Scope: Automated generation of graphical interfaces. In *Proceedings of the 2nd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology (UIST'89)*. ACM, New York, NY, 76–85. DOI: https://doi.org/10.1145/73660.73670

[6] David M. Blei. 2012. Probabilistic topic models. *Communications of the ACM* 55, 4 (2012), 77–84.

[7] François Bodart, Anne-Marie Hennebert, Jean-Marie Leheureux, and Jean Vanderdonckt. 1994. Towards a dynamic strategy for computer-aided visual placement. In *Proceedings of the Workshop on Advanced Visual Interfaces (AVI'94)*. ACM, New York, NY, 78–87. DOI: https://doi.org/10.1145/192309.192328

[8] Marvin M. Chun and Yuhong Jiang. 1998. Contextual cueing: Implicit learning and memory of visual context guides spatial attention. *Cognitive Psychology* 36, 1 (1998), 28–71.

[9] Marvin M. Chun and Yuhong Jiang. 1999. Top-down attentional guidance based on implicit learning of visual covariation. *Psychological Science* 10, 4 (1999), 360–365.

[10] Morgan Dixon and James Fogarty. 2010. Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'10)*. ACM, New York, NY, 1525–1534. DOI: https://doi.org/10.1145/1753326.1753554

[11] Peter A. Frensch. 1994. Composition during serial learning: A serial position effect. *Journal of Experimental Psychology Learning Memory and Cognition* 20 (1994), 423–423.

[12] Krzysztof Gajos and Daniel S. Weld. 2004. SUPPLE: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces (IUI'04)*. ACM, New York, NY, 93–100. DOI: https://doi.org/10.1145/964442.964461

[13] Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. Exploring the design space for adaptive graphical user interfaces *(AVI'06)*. ACM, New York, NY, 201–208. DOI: https://doi.org/10.1145/1133265.1133306

[14] Krzysztof Z. Gajos, Jing Jing Long, and Daniel S. Weld. 2006. Automatically generating custom user interfaces for users with physical disabilities. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility (Assets'06)*. ACM, New York, NY, 243–244. DOI: https://doi.org/10.1145/1168987.1169036

[15] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. 2007. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST'07)*. ACM, New York, NY, 231–240. DOI: https://doi.org/10.1145/1294211.1294253

[16] Saul Greenberg and Ian H. Witten. 1985. Adaptive personalized interfaces—A question of viability. *Behaviour and Information Technology* 4, 1 (1985), 31–45. DOI: https://doi.org/10.1080/01449298508901785

[17] Richard N. A. Henson. 1996. Unchained memory: Error patterns rule out chaining models of immediate serial recall. *The Quarterly Journal of Experimental Psychology: Section A* 49, 1 (1996), 80–115.

[18] Helene Intraub and Michael Richardson. 1989. Wide-angle memories of close-up scenes. *Journal of Experimental Psychology: Learning, Memory, and Cognition* 15, 2 (1989), 179.

[19] Laurent Itti and Christof Koch. 2000. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision Research* 40, 10 (2000), 1489–1506.

[20] Christian Janssen, Anette Weisbecker, and Jürgen Ziegler. 1993. Generating user interfaces from data models and dialogue net specifications. In *Proceedings of the INTERACT'93 and CHI'93 Conference on Human Factors in Computing Systems (CHI'93)*. ACM, New York, NY, 418–423. DOI: https://doi.org/10.1145/169059.169335

[21] Jussi P. P. Jokinen, Sayan Sarcar, Antti Oulasvirta, Chaklam Silpasuwanchai, Zhenxin Wang, and Xiangshi Ren. 2017. Modelling learning of new keyboard layouts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI'17)*. ACM, New York, NY, 4203–4215. DOI: https://doi.org/10.1145/3025453.3025580

[22] Andruid Kerne, William A. Hamilton, and Zachary O. Toups. 2012. Culturally based design: Embodying trans-surface interaction in rummy. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW'12)*. ACM, New York, NY, 509–518. DOI: https://doi.org/10.1145/2145204.2145284

[23] Eileen Kowler. 2011. Eye movements: The past 25 years. *Vision Research* 51, 13 (2011), 1457–1483.

[24] Ranjitha Kumar, Arvind Satyanarayan, Cesar Torres, Maxine Lim, Salman Ahmad, Scott R. Klemmer, and Jerry O. Talton. 2013. Webzeitgeist: Design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'13)*. ACM, New York, NY, 3083–3092. DOI: https://doi.org/10.1145/2470654.2466420

[25] 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 (CHI'11)*. ACM, New York, NY, 2197–2206. DOI: https://doi.org/10.1145/1978942.1979262

[26] Talia Lavie and Joachim Meyer. 2010. Benefits and costs of adaptive user interfaces. *International Journal of Human-Computer Studies* 68, 8 (Aug. 2010), 508–524. DOI: https://doi.org/10.1016/j.ijhcs.2010.01.004

[27] Jeffrey Nichols and Tessa Lau. 2008. Mobilization by demonstration: Using traces to re-author existing web sites. In *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI'08)*. ACM, New York, NY, 149–158. DOI: https://doi.org/10.1145/1378773.1378793

[28] Jeffrey Nichols, Brad A. Myers, Michael Higgins, Joseph Hughes, Thomas K. Harris, Roni Rosenfeld, and Mathilde Pignol. 2002. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology (UIST'02)*. ACM, New York, NY, 161–170. DOI: https://doi.org/10.1145/571985.572008

[29] Jeffrey Nichols, Brad A. Myers, and Brandon Rothrock. 2006. UNIFORM: Automatically generating consistent remote control user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'06)*. ACM, New York, NY, 611–620. DOI: https://doi.org/10.1145/1124772.1124865

[30] Jakob Nielsen. 1993. *Usability Engineering*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

[31] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. 2014. Learning layouts for single-page graphic designs. *IEEE Transactions on Visualization and Computer Graphics* 20, 8 (Aug. 2014), 1200–1213. DOI: https://doi.org/10.1109/TVCG.2014.48

[32] 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 (CHI'15)*. ACM, New York, NY, 1221–1224. DOI: https://doi.org/10.1145/2702123.2702149

[33] D. R. Olsen, Jr.1989. A programming language basis for user interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'89)*. ACM, New York, NY, 171–176. DOI: https://doi.org/10.1145/67449.67485

[34] Angel R. Puerta, Henrik Eriksson, John H. Gennari, and Mark A. Musen. 1994. Model-based automated generation of user interfaces. In *Proceedings of the 12h National Conference on Artificial Intelligence (Vol. 1) (AAAI'94)*. American Association for Artificial Intelligence, 471–477. http://dl.acm.org/citation.cfm?id=199288.184583

[35] Keith Rayner. 2009. The 35th Sir Frederick Bartlett Lecture: Eye movements and attention in reading, scene perception, and visual search. *Quarterly Journal of Experimental Psychology* 62, 8 (2009), 1457–1506.

[36] John Rheinfrank and Shelley Evenson. 1996. Design languages. In *Bringing Design to Software*. ACM, 63–85.

[37] Dario D. Salvucci. 2001. An integrated model of eye movements and visual encoding. *Cognitive Systems Research* 1, 4 (Feb. 2001), 201–220. DOI: https://doi.org/10.1016/S1389-0417(00)00015-2

[38] Sayan Sarcar, Jussi Jokinen, Antti Oulasvirta, Xiangshi Ren, Chaklam Silpasuwanchai, and Zhenxin Wang. 2018. Ability-based optimization: Designing smartphone text entry interface for older adults. *IEEE Pervasive Computing* 17 (2018), 15–26.

[39] A. Sears. 1993. Layout appropriateness: A metric for evaluating user interface widget layout. *IEEE Transactions on Software Engineering* 19, 7 (July 1993), 707–719. DOI: https://doi.org/10.1109/32.238571

[40] Andrew Sears, Julie A. Jacko, Josey Chu, and Francisco Moro. 2001. The role of visual search in the design of effective soft keyboards. *Behaviour and Information Technology* 20, 3 (2001), 159–166.

[41] Kashyap Todi, Jussi Jokinen, Kris Luyten, and Antti Oulasvirta. 2018. Familiarisation: Restructuring layouts with visual learning models. In *Proceedings of the 23rd International Conference on Intelligent User Interfaces (IUI'18)*. ACM, New York, NY, 547–558. DOI: https://doi.org/10.1145/3172944.3172949

[42] Kashyap Todi, Daryl Weir, and Antti Oulasvirta. 2016. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems (DIS'16)*. ACM, New York, NY, 543–555. DOI: https://doi.org/10.1145/2901790.2901817

[43] Anne M. Treisman and Garry Gelade. 1980. A feature-integration theory of attention. *Cognitive Psychology* 12, 1 (1980), 97–136.

[44] Martijn Van Welie and Gerrit C. Van der Veer. 2003. Pattern languages in interaction design: Structure and organization. In *Proceedings of Interact*, Vol. 3. 1–5.

[45] Jacob O. Wobbrock, Shaun K. Kane, Krzysztof Z. Gajos, Susumu Harada, and Jon Froehlich. 2011. Ability-based design: Concept, principles and examples. *ACM Transactions on Accessible Computing* 3, 3 (April 2011), Article 9, 27 pages. DOI: https://doi.org/10.1145/1952383.1952384

[46] Tom Yeh, Tsung-Hsiang Chang, and Robert C. Miller. 2009. Sikuli: Using GUI screenshots for search and automation. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology (UIST'09)*. ACM, New York, NY, 183–192. DOI: https://doi.org/10.1145/1622176.1622213

Received May 2018; revised December 2018; accepted December 2018