Project CSE 503

UITalk: Enabling Blind and Visually Impaired Developers to Edit Visual Web Layouts

Venkatesh Potluri (vpotluri), Christine Chen (chenc55), Liang He (lianghe)

Section 1: MOTIVATION

Access to information about user interface layouts is essential to accomplish many tasks, ranging from the ability to better navigate User Interfaces (UIs) to the ability to build one's own representation of ideas (*e.g.,* prototypes), thoughts (*e.g.,* blogs) and online presence (*e.g.,* personal and professional websites). The UI is the first thing smartphone app developers are taught to build [11, 12]. Despite the emphasis on visual interfaces, existing tools for Blind and Visually Impaired (BVI) developers, such as IDEs and screen readers, do not provide the necessary information to build a well-designed visual interface, which conforms to visual design guidelines such as consistent spacing, proper alignment of UI elements, and proper arrangement of layout across screens of different sizes. In other words, though the underlying code (XAML for windows, Android XML for Android, and CSS for web) is inherently accessible, the outputs (or the effects of changing this code) are not accessible to a BVI developer. This leads to an important open question: *How can BVI developers independently make meaningful changes to the visual design of a webpage that conform to design guidelines?*

To answer this question, we:

a) Create touch-based interaction techniques that allow BVI developers to edit the visual design of webpage UIs.
b) Identify a set of common design guidelines and provide a mechanism that checks that the webpage UI changes BVI developers make conform to design guidelines.

1.1: Goals, Contributions, and User Scenario

Our primary goal is to enable BVI developers to *independently* edit webpages without sighted assistance and without breaking visual design guidelines. In doing this, we focus on enabling blind users to edit the CSS corresponding to a webpage. To achieve this goal, we (i) introduce a new way for BVI developers to edit the CSS file (in addition to editing via a code editor) -- touch based interactions with audio feedback on an accessible canvas (tablet); (ii) implement a validator that checks for violations of three design guidelines when the webpage is edited; and (iii) create a server that links the accessible canvas, validator, and the code editor.
For our work, we restrict our investigation and solution to the context of webpage design. We assume that the user of our tool is a BVI developer with web programming knowledge and experience.

The following is an example scenario of how a user might interact with our system: Say Alice is a BVI developer and wishes to edit her webpage. She opens the UI Talk app on her tablet (which displays a representation of her webpage) and opens up the VSCode code editor and enables the UITalk extension (which opens the CSS file for editing). On the tablet, she uses free exploration

(by dragging her finger on the screen) and audio feedback to locate the heading element she wishes to modify. This gives Alice a spatial understanding of the webpage layout. She uses gestures to change the font to *Helvetica*. UITalk rejects this change, because there are more than three types of fonts on the webpage. This means that the CSS file and webpage representation on the tablet are not modified. VSCode announces the specific design guideline violated (*typeface consistency)* and the line number in the CSS file corresponding to the attempted change. With this information, Alice now knows how to make a valid change and can do so via the tablet or the code editor. To make a valid change, Alice now selects *Times* (a font already applied to other elements on the webpage) for this heading. VSCode now announces the font change, the line number that has been updated in the CSS file, and that the change has been accepted. The CSS file and the webpage representation on the tablet are now updated.

Section 2: Related Work

This section introduces prior work as well as current efforts towards improving the accessibility of developer tools and approaches for BVI programmers.

2.1: Programming Environment Accessibility

The general accessibility literature for BVI developers spans IDEs, teaching methodologies, and introducing BVI individuals to programming. Subsequent discussion is restricted to work that assumes prior programming experience.

Audio-based feedback and enhanced code navigation have been the two major approaches to assist BVI developers. For the former, one of the first artifacts of non-visual access to code was Emacspeak [13], where the system introduced several interesting audio-based approaches with respect to providing semantic information about programs. There have been several attempts such as the Wicked Audio Debugger [15], SodBeans [16], and CodeTalk [10] that facilitate debugging through audio feedback. Schanzer *et al*. present a tool that allows users to navigate code at an Abstract Syntax Tree (AST) level [14]. In one of the largest interview studies with blind software developers, none of the participants did UI development [1]. This speaks to the lack of solutions for BVI UI development. Our work addresses this gap; it is one of the very few efforts in addition to the recent effort to enable blind users to edit spatial layout [5]. Unlike [5], the UITalk system leverages a user's *existing* touch screen interaction capabilities and *existing* devices to modify visual web layouts.

2.2: Touch Screen and Gesture Accessibility

Prior work has investigated gestures [6], capacitive widgets [3], tactile toolkits [7], and physical feedback [2] to enable and enhance blind users' interactions with touch screens. The UITalk system provides a set of touch-based interaction techniques to support UI design for BVI developers. SlideRule [8] suggests design principles (*e.g., risk-free exploration* for non-visual touch screen interactions). These design principles have been adapted by mainstream screen readers such as VoiceOver and TalkBack. Kane *et al*. also dive into specific gestures and limitations in touch screen interactions for BVI users [6]. Our work leverages guidelines and limitations suggested in this prior work to design interactions on an accessible canvas—a touch screen tablet—to modify visual parameters of layouts. Our gesture set for spatial layout editing (explained in subsequent parts of the document) is based on key ideas provided by these explorations of screen reader accessibility. Our proposed interactions enable risk-free exploration

and use multi-touch gestures for activating commands—guidelines provided by [8]. UITalk differs from prior touch screen explorations by using a touch screen for webpage development (i.e. modification of webpage elements). UITalk is one of the very few systems, in addition to Blocks4All [19], that uses accessible touch screens as input for programming.

Section 3: TECHNICAL DETAILS

We built a system that enables BVI developers to edit the webpage UI while conforming to visual design guidelines. The system consists of an accessible canvas (on a tablet), a code editor (on a computer), and the UITalk controller. Figure 1 shows an overview of our system: (1) The BVI developer explores the webpage elements on the accessible canvas or edits the CSS file in a code editor through audio feedback; (2) The UITalk controller receives the proposed updates from the accessible canvas or the notification from the code editor; (3) The diff processor in the UITalk controller processes the proposed updates from the canvas by updating the CSS file; (4) The validator checks that the updates conform to the specified design guidelines; (5) If the updates pass the validator, the UITalk controller sends the accessible canvas information to update its representation of the webpage and announces the status of changes in the code editor. If the UI updates violate the guidelines, the updates made to the CSS file are reverted. Each module (accessible canvas, code editor, and the UITalk controller) will be described in the following subsections.

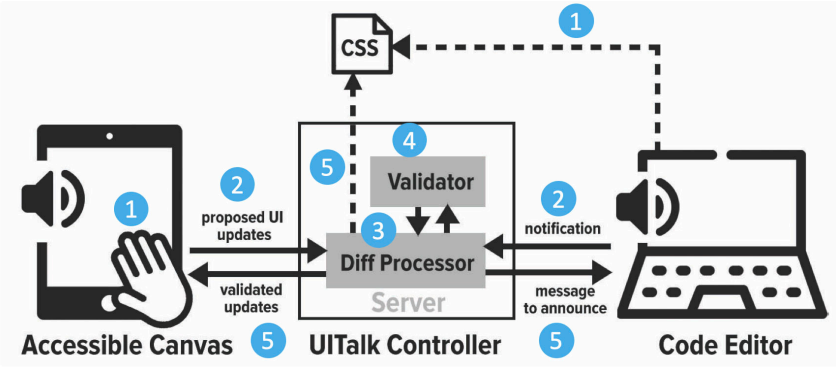

Figure 1. The overview of UITalk system (solid arrows show the data flow and dashed arrows mean actions made to the CSS file).

3.1: Module 1: Accessible Canvas

To allow the BVI developers to touch and manipulate the UI elements, we built the accessible canvas on a touch screen—an iOS app on an iPad. The app displays a representation of the webpage, and the BVI user can perform three tasks through touch and gestures: element navigation, element alignment, and element style property changing. A built-in text-to-speech engine provides audio feedback on the results of the user operations. The gestures are developed with Apple UIKit Gestures[1]. For testing and debugging, a webpage (HTML and CSS files) was manually created. Figure 2 shows the original webpage and the template created. The webpage is

[1] Apple UIKit Gestures:
https://developer.apple.com/documentation/uikit/touches_presses_and_gestures/handling_uikit_gestures

generated following W3C standards[2]. Currently, the following HTML tags are supported: , <h1>, <p>, <a>, , and <img>.

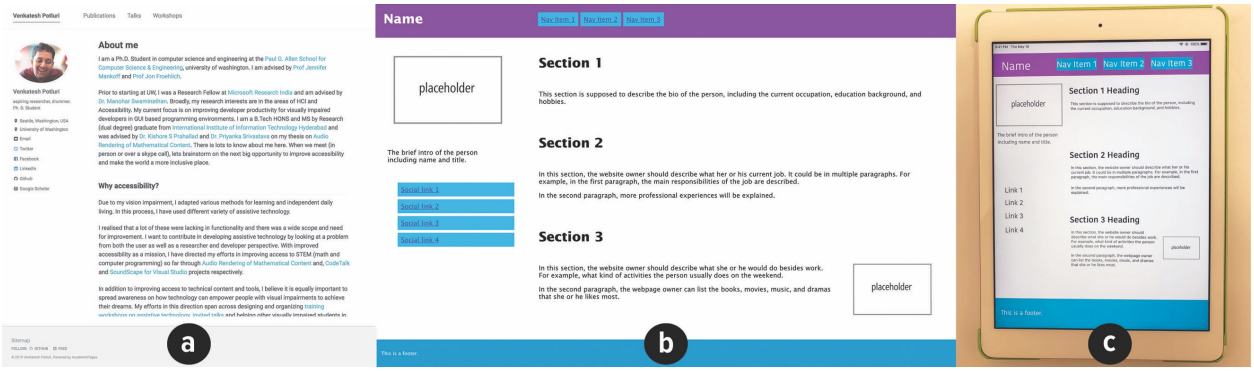

Figure 2. The manually created webpage wireframe (b) based on one team member's personal website (a) is used for debugging and testing. (c) shows the app version of the webpage on an iPad.

3.1.1: UITalk Gestures

A set of touchscreen gestures was created for the following three tasks: navigating HTML elements, aligning HTML elements, and adjusting element style properties. UITalk gestures are created specifically for webpage UI design by combining the standard Apple UIKit gestures (*e.g.,* tap, pinch, swipe, rotate) and the suggestions by [6] (*e.g.,* using the tablet's edge).

For element navigation, the BVI developer taps on the touch screen and the canvas tells the developer which UI element is being touched via audio feedback. Then, the developer can navigate the HTML container structure by dragging the finger around on the screen. To align multiple elements, the BVI developer must select elements first as candidates for alignment by double tapping them. With audio feedback provided by our system, the developer can know and decide if the target element is selected for alignment. After all desired elements are selected, the BVI developer can perform the alignment by using the two-finger swipe gesture. For example, swiping towards the left results in all selected elements aligning to the left. The complete list of gestures for navigation and alignment are shown in Table 1. The BVI developer can switch to making element property changes by executing three-finger triple taps on the touch screen.

Table 1. The gestures for UI element navigation and alignment.

Operation	Gestures	Operation Description
Locating an element	Touch and move	Indicating the currently selected element
Aligning elements	Double taps	Selecting an element that is ready for alignment, reporting all the elements that have been selected for alignment so far
	Two-finger swipe left	Aligning the elements to the left
	Two-finger swipe right	Aligning the elements to the right
	Two-finger swipe up	Aligning the elements to the top

[2] W3C standards: https://www.w3.org/standards/

	Two-finger swipe down	Aligning the elements to the bottom
Element style property adjusting mode	Two-finger triple taps	Entering element property editing mode

Table 2. The gestures for UI element property adjustment (shaded rows indicate mode switching operations)

UI Properties	Gestures (assuming already in element property adjusting mode)	Audio Feedback	Applicable UI Elements
Width	1. Tap any point on either horizontal edge of the tablet 2. One-finger moving horizontally to change the width value	Read out the width	All
Height	1. Tap any point on either vertical edge of the tablet 2. One-finger moving vertically to change the height value	Read out the height	All
Margin/padding Editing Mode	Four-finger long press on the canvas to switch between margin and padding editing	Read out editing mode	All
Margins/paddings	1. One-finger left/up/right/down swipe to indicate the left/top/right/bottom making/padding editing 2. If it is top/bottom margin/padding, one-finger moving vertically to adjust the value 3. If it is left/right margin/padding, one-finger moving horizontally to adjust the value	Read out the value	All
Background Mode	Three-finger long press	Read out editing mode	All
Background color	1. One-finger swipe right to select the next color 2. One-finger swipe left to select the previous color	Read out the color	All excluding & <img>
Foreground Mode	Two-finger long press	Read out editing mode	All
Font color	1. One-finger swipe right to select the next color	Read out the color	All excluding <img>

	2. One-finger swipe left to select the previous color		
Font size	1. One-finger swipe up to increase the font size by 1px 2. One-finger swipe down to decrease the font size by 1px	Read out the font size	All excluding <img>
Typeface	1. Two-finger swipe right to select the next typeface 2. Two-finger swipe left to select the previous typeface	Read out the typeface	All excluding <img>
Submit Changes	Three-finger triple taps		
Clear Changes	Rotate toward a certain angle		

In the UI element property changing mode, the BVI developer can change seven properties of the UI elements using the gestures shown in Table 2. After submitting the most recent edit, the BVI developer can continue changing other properties of the currently selected element in the property changing mode. In the future, we are open to extending the set of gestures to support more properties.

To implement the above gestures, we add one layer on top of the iOS app for touch detection and gesture recognition. Since the UITalk system is not using the built-in voiceover provided by iOS, it maintains a structure of all UI elements of the webpage to support the free exploration of UI elements by the BVI user. When the user touches the screen, the system captures the finger's position and computes which element underneath the touch/gesture detection layer is being touched.

3.2: Module 2: Code Editor

Besides making modifications to UI elements on the accessible canvas, the BVI developer can also directly edit the CSS code in a code editor using a keyboard. The system utilizes Microsoft's Visual Studio Code[3]. While the IDE choice might not be very important for non-BVI developers, it is particularly salient in our case as VS Code is one of the few IDEs that is accessible to BVI developers.

To connect VSCode with the UITalk controller, a custom-made VSCode extension pings the server when the user saves changes to the CSS file. This kicks off the processing described in section 3.3.1. The extension also allows for VSCode to receive messages from the UITalk controller. These messages are displayed as pop-up windows in the editor and are announced by the screen reader.

3.3: Module 3: UITalk Controller

The UITalk controller has two parts: (i) a diff processor that translates proposed updates from the accessible canvas into a format understandable by the code editor and vice versa and

[3] Microsoft Visual Studio Code: https://code.visualstudio.com/

disseminates the validated results; and (ii) a validator that checks if the proposed updates violate design guidelines.

3.3.1: Diff Processor

This section walks through the diff processor functionality.

At a high level:

1. The diff processor receives a proposed change.
2. The working copy of the CSS file is updated.
3. The working copy is evaluated by the validator.
4. If it passes, the working copy is checkpointed, the accessible canvas is updated, and the code editor announces the update.
5. If the working copy is rejected, the working copy gets reverted to the last checkpoint, the accessible canvas is not updated, and the code editor announces the rejection.

The rest of the section describes underlying implementation details. If the proposed changes come from the accessible canvas, they are encapsulated in a JSON package as follows (note that there may be more than one key-value pair representing the proposed updates):

{[html_element_id]@[property_being_modified]: [updated_value]}

A proposed change arriving from the code editor is a message notifying the diff processor that changes have been made to the working copy.

In order for the working copy of the CSS file to be evaluated by the validator, it must be converted into a JSON representation. Note that this is the same data format that the accessible canvas uses to update its webpage representation. Thus, if the validator accepts the changes, this JSON representation is what is sent to update the accessible canvas.

If a proposed change from the accessible canvas is accepted/rejected, the code editor announces a message of the following form:

"UPDATE FROM ACCESSIBLE CANVAS: Attempted to change [property] of [html_element] from [old value] to [new value] at line number [#]. STATUS: [ACCEPTED or REJECTED]. THE FOLLOWING DESIGN GUIDELINES WERE VIOLATED: [Design Guidelines]]"

If a proposed change from the code editor is accepted/rejected, the code editor announces a message of the following form:

"UPDATE FROM CODE EDITOR [ACCEPTED or REJECTED. THE FOLLOWING DESIGN GUIDELINES WERE VIOLATED: [Design Guidelines]]"

3.3.2: Guideline Validator

Once the diff processor updates the working copy (or the file is modified through the code editor), the controller invokes our validator. The validator uses the JSON representation mentioned in the previous section to check if the following design guidelines [18, 4] are violated: typeface consistency, spacing consistency, and color consistency.

Typeface consistency. Too many variations in font types can be distracting, confusing, and borderline annoying. A common recommendation is to use a maximum of three different typefaces [20]. In our approach, the check happens as follows:

- For each HTML element that contains text (e.g., <p>, <h1…h6>, etc.), check typeface.
    - If typeface occurs for the first time, increment counter by one.
- If counter > 3: result is false. Return result. Else: result is true. Return result.

Spacing consistency. Horizontal and vertical rhythms are equally important in a webpage. Varied vertical or horizontal spacing causes visual noise. Consistent spacing between grids (if elements are organized inside grids) creates cleaner content that can be easier to consume. The check proceeds as follows:

- Check horizontal spacing consistency.
    - For each element, get all siblings of that element.
        - If number of siblings (including the current element) < 3: result is true and check proceeds to the next element.
        - Else:
            - If the sum of the right margin of the first element and the left margin of the second element is the same for every two adjacent elements: result is true. Else: result is false. Return false.
- Check vertical spacing consistency.
    - For each element, get all siblings of that element.
        - If number of siblings (including the current element) < 3: result is true and check proceeds to the next element.
        - Else:
            - If the sum of the bottom margin of the first element and the top margin of the second element is the same for every two adjacent elements: result is true. Else: result is false. Return false.
- If result is true for both horizonal spacing consistency and vertical spacing consistency: return true [guideline is not violated]. Else: result is false. Return false [guideline is violated].

Color consistency. Color is used to group related items and similar colors infer a similarity among objects. UI elements should share the same color scheme, for example, all hyperlinks should have the same color throughout the webpage.

The color consistency check happens as follows:

- For each element that contains text, check the type and color.
    - If the color of a particular type is different from what has been seen before for the said type, result is false and stop check. Else: result is true and proceed to the next element.

- Return result.

Section 4: EVALUATION AND RESULTS

A preliminary evaluation of our prototype with an external participant was performed. This evaluation consisted of 3 parts: evaluation of participant's current webpage development strategies, evaluation of participant's use of UITalk system, and post-study interview. The participant is congenitally blind with no light perception. He is a professional software engineer and also has his own website. The participant's responses were audio recorded with his consent, and one member of the research team took notes during the study session.

Section 4.1: Evaluating Current Webpage Development Strategies

The goal of this section was to understand the current strategies our participant follows to design and develop a webpage. The following questions were asked:

- Did you design any webpage?
- How did you go about making design decisions on how the webpage should look?
- For the webpages you built, what tools did you use?
- When was the last time you updated a webpage?
- What were the updates about?
- How did you update the website? What tools did you use to update your website?

Following this, the participant was presented with a blank tablet and asked to explain the layout of his website. This was necessary to observe the approach the participant takes to explain his webpage without any tactile or audio feedback. These observations were used to better explain the template to users.

Section 4.2: UITalk Introduction and Testing

In the second part, the participant was introduced to the UITalk system. First, the webpage template was explained. Then, the participant was allowed to freely explore the template by touch. After the free exploration, the participant was asked to describe his understanding of the layout of the webpage. Note, he had audio feedback provided by the system.

After the introduction session, the participant performed three tasks (Table 3).

Table 3. Assigned tasks in the user evaluation

Task	Expected Outcome
1. The participant is asked to change the typeface of the bio paragraph.	Three typefaces have been used on the webpage: *Helvetica*, *Times*, and *Arial*. The typeface of the bio paragraph uses *Times*. If the participant changes that typeface to either *Helvetica* or *Arial*, the change would not break the design guideline. Otherwise, the typeface consistency guideline would be violated.
2. The participant is asked to increase the bottom margin of the second hyperlink in the left sidebar of the webpage.	This change will lead to a violation of spacing consistency. The participant will need to change the bottom margins of the other three hyperlinks to make the vertical spacing consistent.

3. The participant is asked to left align the image and the bio in the left sidebar.	Change Accepted.

Section 4.3: Post-Study Interview

After the task session, the participant was asked to rate the system according to the system usability scale [17] as well as answer the following questions:

- Did the system provide enough information for completing these tasks?
- What more information did you expect the expect had given you?
- What was easy?
- What was confusing?
- What features should be added?

Section 4.4: Results

Web Development Strategies: The participant used his friend's template for his website and performed content-based edits using a plaintext editor (Notepad). The participant also provided insight about his strategy for template selection:

> *"I try to pick templates that are more flexible than others, so that even …if I add more content, the likelihood of things messing [up] the layout is low."*

For subsequent edits, the participant had the changes checked by a sighted friend or colleague. In cases where the updates were just about adding and linking to a file, the participant was able to make the necessary changes independently.

> "*And sometimes because it's just a personal website, I take the risk. If I just want to upload a document, for example, for someone to see, I'll just upload it and not really worry that much about how it is appearing because often times it's just going to be a page of its own.*"

UITalk Tasks: The participant completed all the three tasks successfully with some prompts from the team (in terms of gestures). Towards the later part of the study, the participant could perform the gestures with less assistance.

Our participant had some difficulty in accurately performing some gestures. However, on testing the gestures with the BVI developer on our team, the observed success rate was higher (the success rate was not quantified). Earlier research [8] shows that there needs to be an accuracy tradeoff when designing touch screen interactions for BVI users. However, existing gesture recognition APIs did not allow for this. While existing screen readers do accommodate for accuracy errors, there is no information on how they do this.

Post-Study Interview: Table 4 presents the results of the SUS questionnaire. Note that the sample size is 1 and, thus, these results may not be conclusive of the general effectiveness/usability of our approach.

Table 4. The result of the SUS questionnaire (1: strongly disagree; 5: strongly agree)

I think that I would like to use the system frequently.	3
I found the system unnecessarily complex.	2
I thought the system was easy to use.	3
I think that I would need the support of a technical person to be able to use the system.	1
I found the various functions of the system were well integrated.	1
I thought there was too much inconsistency in the system.	1
I would imagine that most people would learn to use the system very quickly.	5 (in terms of gestures)
I found the system very cumbersome to use.	2
I felt very confident using the system.	2
I needed to learn a lot of things before I could get going with the system.	1

In terms of the information provided by the system, the participant said he would have preferred to have access to a list of changes (and be able to revert specific changes). The participant also wanted the tool to give a better sense of the overall layout of the page. The participant was not convinced that touch was the most effective means of making modifications and suggested using alternative input-output techniques (e.g. voice interactions *and* the existing screen reader).

Section 5: Limitations

Our work has limitations with regards to scalability, verification of the layout changes, and interactions on accessible canvas.

Section 5.1: Scalability and Supporting Modern Development Workflows

For this project, we assume the CSS file is formatted in a very particular way in order to facilitate processing. We only support changes to the CSS file—we do not support creation or deletion of HTML elements.

Currently, our validator does not check for accessibility guidelines like VizAssert [9] does. Future versions of our validator can be expanded to check and ensure that user edits conform to accessibility guidelines. Another limitation is the way we must "rebuild" webpages in order to present them on the accessible canvas. While it would be ideal for our system to work with a webpage open on a tablet browser, current limitations with screen readers as well as limitations in the touch screen gesture APIs that operating systems expose (when accessibility services are

enabled) prevent us from achieving this. Further exploration into the way NVDA[4] handles touch screen interactions may enable us to achieve this.

Section 5.2: Verification of Edits

Our goal is to use formal verification techniques to verify that edits to visual layouts do not break visual design guidelines. For this, we originally intended to use VizAssert. However, from our investigation thus far, the VizAssert processing delay results in an experience that is far from real-time. Instead, we have had to implement our own validator (note that our validator does not check for the accessibility guideline violations that VizAssert checks for).

Section 5.3: Interactions with Accessible Canvas

It would be ideal for the accessible canvas to be able to interact with existing screen readers. However, existing implementation of VoiceOver and the APIs Apple provides make it infeasible to build apps with custom gestures that can function alongside VoiceOver. We have implemented a subset of functionality that VoiceOver offers and acknowledge that it will take some adjustment for screen reader users to work with our system. In addition, the learning curve increases as more gestures are introduced to the BVI developer.

Section 6: Discussion and Future Work

Our preliminary investigation in this space opens up interesting research questions in the context of verification, interaction, and skill.

6.1 Verification

It is worthwhile to investigate the best strategy to provide precise feedback on violations in real-time. Existing approaches, such as VizAssert, that depend on formal verification do not seem to be able to provide this real time experience.

6.2 Interactions and feedback

In this work, we explored a multi-modal (keyboard typing, touch/gesture, and audio) approach to enable BVI developers to edit spatial UI without breaking visual design guidelines. Our decision to use a touch screen tablet (accessible canvas) and an IDE on a desktop were drawn from a currently unpublished study where we observed that BVI technology users had better perception of the visual layout of apps they used on touchscreen devices in comparison to webpages accessed through a desktop computer. Moreover, our current approach purely relies on speech feedback. The potential use of non-speech audio cues and spatial audio (when the developer wears headphones and it seems as though sound is coming from different directions) remains unexplored.

6.3 Existing skillset

Existing research shows that BVI developers do not currently work on front-end technologies. There needs to be research on whether skilled BVI developers that are well acquainted with knowledge about CSS exist. If not, can approaches such as those proposed in our system be used to teach CSS?

---

[4] https://nvaccess.org

Section 7: REFERENCES

1. Albusays, K., Ludi, S., & Huenerfauth, M. (2017, October). Interviews and Observation of Blind Software Developers at Work to Understand Code Navigation Challenges. In *Proceedings of the 19th International ACM SIGACCESS Conference on Computers and Accessibility* (pp. 91-100). ACM.
2. Bau, O., Poupyrev, I., Israr, A., & Harrison, C. (2010, October). TeslaTouch: electrovibration for touch surfaces. In Proceedings of the 23nd annual ACM symposium on User interface software and technology (pp. 283-292). ACM.
3. Chan, L., Müller, S., Roudaut, A., & Baudisch, P. (2012, May). CapStones and ZebraWidgets: sensing stacks of building blocks, dials and sliders on capacitive touch screens. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 2189-2192). ACM.
4. Leavitt, M. O., & Shneiderman, B. (2006). based web design & usability guidelines. Background and Methodology.
5. Jingyi Li, Son Kim, Joshua A. Miele, Maneesh Agrawala, Sean Follmer. 2019. Editing Spatial Layouts through Tactile Templates for People with Visual Impairments. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19). ACM, New York, NY, USA. DOI: https://doi.org/10.1145/3290605.3300436
6. Kane, S. K., Wobbrock, J. O., & Ladner, R. E. (2011, May). Usable gestures for blind people: understanding preference and performance. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 413-422). ACM.
7. Kane, S. K., Morris, M. R., & Wobbrock, J. O. (2013, October). Touchplates: low-cost tactile overlays for visually impaired touch screen users. In Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (p. 22). ACM.
8. Kane, S. K., Bigham, J. P., & Wobbrock, J. O. (2008, October). Slide rule: making mobile touch screens accessible to blind people using multi-touch interaction techniques. In *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility* (pp. 73-80). ACM.
9. Panchekha, P., Geller, A. T., Ernst, M. D., Tatlock, Z., & Kamil, S. (2018, June). Verifying that web pages have accessible layout. In Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 1-14). ACM.
10. Potluri, V., Vaithilingam, P., Iyengar, S., Vidya, Y., Swaminathan, M., & Srinivasa, G. (2018, April). CodeTalk: Improving Programming Environment Accessibility for Visually Impaired Developers. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (p. 618). ACM.
11. Build a simple user interface. Android Developers. (n.d.). Retrieved from https://developer.android.com/training/basics/firstapp/building-ui.html
12. R., Wenderlich. (2018, July 24). Your First iOS App · Challenge: Making a Programming To-Do List. Retrieved February 24, 2019, from https://www.raywenderlich.com/5993-your-first-ios-app/lessons/2
13. Raman, T. V. (1996, April). Emacspeak-direct speech access. In *International ACM Conference on Assistive Technologies* (pp. 32-36).

14. Schanzer, E., Bahram, S., & Krishnamurthi, S. (2019, February). Accessible AST-Based Programming for Visually-Impaired Programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 773-779). ACM.
15. Stefik, A., Alexander, R., Patterson, R., & Brown, J. (2007, June). WAD: A feasibility study using the wicked audio debugger. In *15th IEEE International Conference on Program Comprehension (ICPC'07)* (pp. 69-80). IEEE.
16. Stefik, A., Haywood, A., Mansoor, S., Dunda, B., & Garcia, D. (2009, May). Sodbeans. In *2009 IEEE 17th International Conference on Program Comprehension* (pp. 293-294). IEEE.
17. System usability scale: https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html
18. Web design principles: https://theblog.adobe.com/12-dos-donts-web-design-2/
19. Lauren R. Milne and Richard E. Ladner. 2018. Blocks4All: Overcoming Accessibility Barriers to Blocks Programming for Children with Visual Impairments. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (CHI '18). ACM, New York, NY, USA, Paper 69, 10 pages. DOI: https://doi.org/10.1145/3173574.3173643
20. Typography Elements Everyone Needs to Understand: https://bit.ly/2LOMQSL