Software Engineering Assignment 1: Development Difficulties

Venkatesh Potluri – vpotluri

Time taken: 4 Hours

Difficulty 1: Designing User Interfaces non-visually is not possible

In my experience as a developer, it seemed like being able to design interfaces is a very important skill to have. To me, it all started when I wanted to learn iOS and android programming; the first thing any online tutorial starts with is to build a UI. This was a huge barrier to me; there seemed to be a few undocumented ways of somewhat getting it right, but the only way for me to get access to these workarounds was to send an email to a blind Mac user list and wait for a day or two for somebody to respond. There are many glaring issues in using the existing developer tools; even if I managed to get the User Interface (UI) element on to the canvas, the screen reader wouldn't announce if the element is out of the viewport. Often, another bit of missing information is overlapping elements. For example, The only changes I made from a perfect looking version of my personal website were add and remove some text. As a result, there was overlapping text on my website. Screen readers do not announce such overlaps as these, and as a blind developer, I wouldn't know how to fix them either or whether they are fixed. Clearly, a blind software developer should be able to fix their own webpages confidently!

Existing Solutions

Some tools like Apple's XCode provide an alternative tree view of the interface. However, this does not provide the information on whether the element is in the viewport, orientation and size. There is very little work done in this space. (li et al, 2019) propose a system with tactile overlays to enable Blind and Visually Impaired (BVI) users to build layouts. Among the gamut of accessibility-related challenges BVI developers face, enabling non-visual ways to build visual layouts has garnered very little attention.

Proposed Solution

While I do not have a concrete way of solving this, scoping this down to building web layouts may be a good starting point. The first challenge is to even decide on a baseline layout or template to build upon. One way we could do this is to come up with descriptions with a familiar template for a user to base upon. For instance, "The template looks like your Facebook profile" or "the template looks like the CNN homepage". Moreover, smartphone screen readers like Apple's Voiceover and Google's Talkback offer the capability for blind users to directly manipulate UI elements, offering the ability to get spatial information as a result. (Kane et al, 2008) and (Kane et al, 2011) give more insight into this observation. Another approach to support designing layouts could be to use a multi-device approach; a tablet could be used as a physical canvas, and the desktop could be used to do the programming part of designing an interface. (Panchekha et al, 2018)'s viz-assert's visual language could facilitate verification of layouts.

Difficulty 2: Fully-automated comprehensive accessibility testing is not possible

This stems from my experience as a software developer intern for the team responsible for Microsoft's Office accessibility on Android. The automated tests we had in place weren't comprehensive enough, and the tester we hired could not catch all bugs. Moreover, Google would often update Talkback in the background introducing new regressions, which I often discovered just before release or many versions later. The problem with discovering just before release is that the fix wouldn't be ready in time and the feature or code change corresponding to another bug fix would have to be pushed back until the next release cycle. The problem with discovering these much later is that we would not know which version update of Talkback caused the issue. I was in situations where I had to install every version of Talkback to trace the regression and investigate it. Moreover, I also felt that the tester did not have the necessary experience to identify some of the bugs I as a screen reader user found. Also, as a developer, I think it is not my job to do accessibility testing. I mean, being the only blind person in the entire team, I was burdened with accessibility test requests from many teams too. Some organizations mitigate the expertise problem by hiring BVI individuals to accessibility test. While this is an employment opportunity, many disability-related employment efforts convince themselves of doing a great job by placing a majority of BVI individuals in these tester jobs irrespective of their skillset.

Existing Solutions

There are many tools like Microsoft's accessibility insights, Google's chrome dev tools, and android accessibility scanner with varying capabilities but none of them are capable of a comprehensive accessibility test. For example, these tools can tell if there is alt-text but can't identify whether the alt-text is meaningful.

Proposed Solutions

Here is a brainstorm of a few approaches that could be used to identify accessibility issues. First, it is important to capture the semantics of accessibility in apps. For instance, crowd powered approaches, in combination with image description APIs could be used to identify relevance of alt-text. Here is a rough 2- staged approach in doing this: First, the element needing alt text E.G. an image can be sent to an image description API like Microsoft Cognitive Services or Google Cloud Vision. If there is a mismatch, it could be flagged as an accessibility error. If the match doesn't result in a confident result, the image and the caption could be sent to a crowd sourcing service like amazon's mechanical tirk and could be verified for relevance. For deeper accessibility tests requiring screen reader expertise, these could be broken down into micro tasks and can be used in place of captcha for verification. I am sure screen reader users would be happy to do such tasks instead of listening to unintelligible speech and verifying that they are not robots.

Difficulty 3: Collaborative coding activities are not accessible

It is common for teams across geographical boundaries to collaborate on code for code reviews, walkthroughs or knowledge transfer. For a sighted developer, this is straightforward; connect via Skype or Google Hangouts and share screen. But everything that happens in a shared screen is inaccessible to a developer using a screen reader because the shared screen is a video. In addition, collaborative programming tasks like group code reviews become accessible as well because the text that is projected on screen remains inaccessible to a BVI developer using a screen reader. I have faced this in several scenarios: group code reviews while I was a software developer intern, knowledge transfer from a team not geographically co located. I remember the second experience very well; the engineer on the other end was very willing to help me understand the code but when I told them my screen shares wouldn’t work, the depth of information they could give ended up becoming very limited.

Existing Solutions

A lot of accessibility research on developer tools has been focused on making Integrated Development Environments (IDEs) accessible. As far as I know, and I could check, in-accessibilities in the collaborative coding space have not been tackled.

Proposed Solutions

To make collaborative programming accessible, it is very important to situate a BVI developer in the programming environment of their other colleagues. Moreover, rich screen reader specific ide features should be supported. One possible approach is to build on top of Microsoft’s VisualStudio Code IDE, using their live share API. This coupled with screen reader specific enhancements like a mechanism to keep track of multiple cursors could result in a rich collaborative programming experience.