TEACH ACCESS REPORT
MICHAEL MCQUAID
JUNE 2022

INTRODUCTION

This report documents my Teach Access project of 2021–22. Most readers can skip directly to the sections called Viewing the Slides and Viewing the Videos and use the links there to view the associated slideshows and videos. The remainder of the report explains the background of the project, the method(s) I've used and the results obtained and is mostly of interest to lecturers who wish to modify the slideshows.

BACKGROUND

In my proposal to the Teach Access project, I explained that *I have awakened to the mission of Teach Access over the past 5 years of coordinating the Human Centered Computing (HCC) program in the School of Information at the Rochester Institute of Technology (RIT). We have untapped opportunities to work with industry and disability advocates to enhance student understanding of digital accessibility.*

I went on to say that *when I came to this position, I was under the mistaken impression that RIT had solved the accessibility problem by having a required course in accessibility. I did not realize that students pigeonholed accessibility into a* ***category*** *of specialization. Students think of accessibility as one of several categories in which they can specialize and see the course on accessibility as irrelevant to mainstream design. Since then, I have come to recognize that we need to integrate accessibility into the curriculum rather than standing it apart in its own niche.*

My proposal was to add an accessibility module to my Prototyping and Usability Testing course. I said that *I designed this*

course without reference to accessibility because there was already a required course in accessibility and there was so much to cover. I wrongly believed that students would receive all the knowledge they needed from the required course in accessibility.

I justified this by my own experience, saying that *since then, I have begun to conduct research in accessibility and have had a couple of papers in CHI and ASSETS on the subject. I'm working with students in the research and they are hard to find. Why? Because most of the students consider accessibility as a specialization and not as part of the fabric of computing. I see the need to rethink how we integrate accessibility into the curriculum. I can now see that it needs to be woven into the fabric of each course.*

My plan was expressed in three parts, as follows. *How can I best accomplish my new goal of bringing accessibility into my introductory course? I need time to plan, participation from disability advocates, and materials in the form of software and books. TeachAccess could enable all three of these things through the TeachAccess grant. If my application is accepted, I plan to use the funds for those three things, (1) to pay for summer support for planning and organizing, (2) to pay a stipend to a disability advocate to appear as a guest lecturer, and (3) to purchase materials including software and books.*

The module I intend to add is based on the Teach Access set of fundamental concepts and skills needed to implement inclusive design. This module would occupy Week 2 in the attached syllabus. The following topics would be moved forward and the current Week 8 topic, Teamwork, would be eliminated, as that is well covered in the Senior Development course and is not mentioned in the Course Proposal Form for this course, thus requiring no oversight to remove it.

This module, like all my materials, will be available to future teachers of this course, as well as graduate equivalents of the course. When it becomes mature enough, it will be added to my

public website at https://mickmcquaid.com*, where I have included content from some of my other courses that may be freely used by other educators. Such content is also available on my Github site, where it may be freely modified by other educators. I believe Teach Access can make progress by encouraging the use of public, flexible repositories like Github for educational material, as well as by encouraging the use of flexible licenses like the MIT license for remixing and reusing such material.*

I had to revise this plan over time, as I will describe in the next section.

METHOD

My main method has been to develop annotated slide shows in an accessible format and place them on Github in a format that is free to use and easy for others to modify. In addition to these slideshows, I have recorded two lectures by blind advocates for accessible computing. These lectures are available on Youtube without advertising.

Each slideshow consists of two parts, a plain text markdown file and a set of image files linked to and explained in the markdown file. Next I will describe how these files are created and can be edited.

Slideshow software. Initially, I planned to use slideshow software called *Xaringan*. I abandoned this plan after creating the first slideshow, as will be explained in the Results section below. The rest of this section concerns the software I then turned to.

The slideshow software is called `reveal.js` and is a free to use open source JavaScript library for creating slideshows in html format. This format was selected because it is inherently more accessible than proprietary slideshow software. Proprietary software must be purchased and is not always available

and the techniques for making proprietary slideshow software accessible must be repeated every time the slideshow is updated. The `reveal.js` library acts on an html file and not everyone can write html, so a simpler format is used to create source files in markdown format (to be explained next) that is then converted to html format by another free open source software tool, `pandoc`. The `reveal.js` and `pandoc` steps are only necessary for those wishing to contribute to the project by modifying slides and can be skipped by end users, who can just open the html file in any contemporary browser. The image files must reside in the same directory as the html file if the end user chooses to download the package. Otherwise, the html file resides on my personal website and will remain there indefinitely as I plan to use it in future courses.

Markdown. Each lecture is written in a format called markdown, using a text editor, such as Notepad++ on Windows, TextEdit on macOS, or gedit on Linux. Markdown was originally invented as a simplified replacement for html for bloggers. It has since attracted a large following due to its simplicity and a wide variety of software has emerged to convert it to html, word processors, pdf, and other formats. `pandoc` is its most popular free open source converter.

Markdown involves simple tags, simpler than html. For example, a first level heading is just the text preceded by a hashtag symbol. A second level heading is the same, but preceded by two hashtag symbols. Surrounding a word with asterisks puts it in italics. Surrounding it with a pair of underscores makes it bold face. An image file can be added to the file with the construct

```
![alt text](filename)
```

presuming that the filename is a reachable file, either in the current directory or at a named path.

A link can be added with the construct

`[text to be displayed](URL)`

where the text to be displayed will be highlighted according to a default rule or a rule encoded by the author, using CSS or any of a number of other formatting tools. In the case of these lectures, I have formatted everything according to my preferences, including the colors and typefaces (which are automatically downloaded from Google when the slideshow is rendered) so that the end user need only know how to create formatting rules if they desire to modify the look and feel of the slideshow.

For those who wish to modify the slideshows, there are extensive tutorials for markdown available on the web, such as pandoc markdown for the specific version of markdown being used in these slides, or markdown tutorial for an interactive introduction to plain vanilla markdown, without any of the `pandoc` extensions.

Github. Github is the most popular public repository for the free open source version control software known as `git`. The `git` software allows people to easily collaborate on mostly software projects, but also on the authorship of documents, such as reference manuals and slideshows. There are literally thousands of slideshows available on Github. (Unfortunately, the Github company was acquired by Microsoft, which previously evinced hostility to free open source software. So far, Microsoft has not tampered with Github but, if it does, I plan to move to a compatible public repository such as Gitlab.)

Github allows you to check out a version of a file from a repository, modify it on your own machine, then check it back in for your collaborators to use. If two people check out the same file at the same time and modify it in two different ways, there is a protocol for merging the results. All of the slideshow files for all the lectures, along with all there image files and formatting information, are on Github now, available

to any wishing to modify or otherwise use them.

The `git` software supports two important processes relevant to this project. One is called a *pull request*, where a contributor asks the owner of the repository (me) to accept changes to a file or files. This is the mechanism whereby someone can contribute to this project by improving the slideshows.

The second important process supported by `git` is called *forking* and this is something that can be done when a contributor decides to go their own way and modify the slideshows in a way that is not supported by the repository owner (me). This allows someone with a different viewpoint to edit the slideshows as their own project, diverging from the original project but using the original project as a starting point. Why would a repository owner want to allow this? There is usually more than one way to do things and each repository owner must choose their one preferred way. You can't be all things to all people. Many healthy projects have engendered forks where differences in philosophy have led groups of contributors to split up and offer software (or documentation or books or whatever can be contained in files) that satisfies two different communities with different needs.

RESULTS

The main products of this project are the accessibility slides in two forms and the video lectures.

At the time of this writing, some of the slides are not in their final form. This is because I began using a different slideshow software that I abandoned midway through the project. The instructions for its use differ slightly from what is described below. Pressing the `?` key during viewing of the slides reveals the differences between the two formats. The *Xaringan* software that I began the project with requires more software

expertise and the installation of software on the local computer of the lecturer modifying the slides, unlike `reveal.js`, which allows either (1) direct modification of the html files without installing any particular software other than a text editor (which is usually preinstalled on most computers), or (2) direct modification of the markdown files, which only requires the installation of `pandoc` on the local computer. In either case, using the slideshows without modification only requires a web browser.

The remainder of this section describes the outcome as if all the slides have been converted from *Xaringan* to `reveal.js`, since that is the anticipated final outcome of this project. Anyone using the slideshows soon after this report is filed will notice some discrepancies that will eventually vanish. For example, to obtain speaker notes in *Xaringan*, the lecturer presses **c** on the keyboard, where the letter c stands for Clone, meaning to clone the slideshow to a second window. There is no counterpart to this in `reveal.js` wherein either of two windows can be designated as speaker notes.

My personal website. The slideshows are viewable on my personal website and can be downloaded from there.

The link is iste264 which is the RIT code for the course in which these slideshows constitute a module. The individual slideshows are available at

- 01understanding
- 02context
- 03uiFacilitatorsBarriers
- 04assistiveTech
- 05bestPractices
- 06appliedTechniques

Viewing the slides. For a lecturer using the slides, there should be two copies of the slides opened in separate windows

of the same web browser. In one of the windows, which may be connected to an external monitor such as a lecture hall projector, the lecturer should press the letter **f** on the keyboard. This stands for fullscreen mode and puts that copy of the slideshow into presentation format. On the other window, which may be displayed on the lecturer's laptop screen, the lecturer should press the letter **s** on the keyboard, which stands for speaker notes. The speaker notes contain both commentary and the sources for the images. The full screen presentation and the speaker notes screen are automatically synchronized so that advancing one advances the other.

The lecturer can press the `?` key on the keyboard to see what else is supported, such as a timer, screen blanking, and different kinds of navigation.

One oddity of the `reveal.js` software is how it handles the slideshow overview mode. Most slideshow software just arranges small images of slides in a rectangular grid in this mode. Instead, `reveal.js` offers a row of slides marked as headings across the top and vertical columns of subsidiary slides beneath these. I find that the easiest way to navigate a slideshow is to repeatedly press the spacebar. This advances to the next slide in the current column until it reaches the end of the column, then advances to the next column. If the lecturer loses their place, they can press the **o** key to see the slides laid out in rows and columns and can use the arrow keys to navigate.

Modifying the slides. To modify the slides, the best practice is to use `git` so that the modifications can be added to the Github repository. There is nothing stopping a lecturer from simply modifying the slides without contributing to the original project, however. If this course of action is desired by a lecturer, the simplest approach is to visit github and download the slides as a zip file. Unzip the file and modify the slides

as desired.

To follow the best practice approach instead, installing `git` on your local machine is the first step. There are many online tutorials for doing so and they vary according to your operating system. For example, on my macOS system, I visited homebrew and followed the instructions there to install *Homebrew*. Then, I said

```
brew install git
```

in a terminal window to install git. Similar instructions may work for Windows or Linux but will vary according to the lecturer's local setup.

The second step is to clone the repository. This can be done in a terminal window by saying

```
git clone https://github.com/mickmcq/accessibilitySlides
```

Now, the lecturer is in a position to contribute to the original project by submitting pull requests or to set out on their own path by forking the project. Both these activities will require the lecturer to follow a `git` tutorial or otherwise become familiar with `git`.

Viewing the videos.

- Kim Kline
- Catherine Lewis
- Kiki Smith

The above links are to Youtube, where the videos will remain permanently. The Kiki Smith talk was by far the best received by students.

CONCLUSION

This remains an ongoing project to integrate accessibility into the HCI curriculum, particularly in the context of the creation and testing of design prototypes. The materials will continue to be improved by the author and perhaps by others who may feel motivated to contribute!