Capstone Experience:
Tutor Scheduling

Dan Coleman

4 May 2016

This project was one born out of a desire for the automation of common processes. One of the main strategies for collecting availability data is to send out a link to an Excel document and ask the people being scheduled, tutors in this case, to fill out the cells corresponding to their preferred times and the times they are busy. The administrating professor then, either by hand or multi-step program, proceeds to schedule the tutors for the course.

This scheduling program has two points of contact with the professor under "normal" program operation: It takes each tutor's personal information and the basic course info from the professor, and it requires the professor to press a button to actually make a new schedule. Besides that, and emailing the tutors a link to the schedule, the professor does not interact with the schedule at all. Thus, the automation has been greatly increased, bearing a large portion of the time constraint previously placed on the professor.

To begin the project, a conversational approach to collecting the project requirements continued to the point where a formal requirements document could be authored. This led to a construction of a general flow diagram for the system, including as many features of the system as could be thought of. The general flow model, in addition to the informal requirements, naturally led to the creation of the project's specifications. As with most development cycles, the ideas presented in the specifications led to an enumeration of the various actions a user of the system has the potential to take. Different actions require different features, and, as always, some are more relevant than others, leading to the necessity of a cohesive design, which was ultimately constructed with those in mind. While I had originally thought I would be able to develop the documentation prior to actually implementing any source code, the broad design framework was best visualized in code.

Once I began implementing the code, the basic HTML outline within the PHP code was created first. Gradually, the PHP functionality was increased to include databases and the use of sessions.

One of the challenges encountered at roughly this point in the development process was database connectivity. At the beginning of the project, the primary development apparatus was on my personal machine. However, due to an unforeseen computer malfunction, I was forced to switch development to the sand server. At that point, though, I did not have a database set up for use with the scheduling system. Setting this up delayed progress by a couple of valuable days, which,

while unfortunate, caused no major damage because the entire system was, and still is, tracked with a GitHub repository. The lesson was clear: Always have a reliable backup strategy in place.

From this point, functionality was expanded to include JavaScript button handling. At first, the JS was strictly responsible for performing actions when the buttons were clicked. Because certain event handlers were not working correctly with this philosophy, my approach changed to force JS to create the buttons at load time. Doing so made attaching the event handlers very simple. Knowing how to do this was important again later, when other buttons required their own functional attachments.

Following successful button operations, I worked to implement the scheduling process itself. Understanding the output was immediately important, so I wrote the HTML output writer shortly after. Once the basic syntax errors were fixed, the system was able to run data through the scheduler, but the output was either incorrect or non-existent. It turned out the portion of code that assigned hours to tutors was not conducting a precondition check as intended, resulting in the assignment of more tutors to a given hour than was allowed. It was as though the program was polling a tutor object for its preferred hours and automatically assigning it those hours. Thus, highly desirable hours had many tutors tutoring, and busy hours had no one signed up for tutoring. In effect, the needed precondition check flattened the distribution of times throughout the week to the point where each shift only had the requisite number of tutors assigned. Originally, the recommended number of tutors per shift was two tutors, but with the aid of data generated later, it was determined that a more successful model would be if the administrative entity was given the ability to put how many tutors he would like to have tutoring per shift. Therefore, the schedule now writes with a statistic at the bottom of the schedule page telling what percentage of tutors have complete schedules. By "complete schedules," it is meant schedules that assign each tutor the number of hours he was cleared to work.

So, unless the number of tutors per shift is changed, the system will not always have each tutor scheduled for the right amount of hours, but it will have a tutor scheduled for each tutoring hour. Another shortcoming is that the system assumes a tutor is only tutoring for one course. If there is a person tutoring a Calculus course and simultaneously tutoring a Discrete Mathematics course, the system does not know how to work with that. However, requirements for the system only dictated

that the system should work with one course, so dealing with tutors tutoring multiple courses is beyond the immediate scope of the project.

In a possible future version of the project, having a schedule page that is able to be changed would be highly beneficial. One thought I had would be to change the structure of the output page to Sortable lists of Sortable lists using the Scriptaculous library. That would enable the administrating professor to simply drag the entries in the schedule to achieve a better, more complete schedule. Another addition that would make correcting any errors easier would be adding a page that lets the administrator view each tutors preferred times and class times. Such an ability would make rescheduling possible without consulting the database or the tutor himself. Right now, the system is concerned with processing in the relevant course-related data and outputting a schedule that incorporates as many of the tutors as is feasible. Also, in a later version, it would be advantageous to incorporate each individual's education standing, i.e., graduate or undergraduate, as it is generally accepted that graduate students have a higher priority regarding shift availability.

For my capstone project, I worked alone throughout the development process, even though it was not a particularly long development process. In my case, the physical condition of the development process was rather stressful, both mentally and physically, with a complete project time-line of roughly six weeks. I do not think the development experience would have been as bad had the process been more drawn out, but such was not the case. During the project, the interactions between myself and professors were, due to the cramped nature of the time-line, decidedly limited.

Originally, I was developing on an Intel-based Mac, using the text editor TextMate and running command line segments in Terminal. After the switch to sand, a Ubuntu Linux server, the code transferred fairly directly to where most of my practices were the same either way. I still ran command line segments in a terminal window and used Google Chrome to develop the web pages. Of course, I no longer used the Mac-based editor and instead used the less-advanced editor gedit. Over both systems, I used Git version control software to track my progress and keep an accurate backup of my files. For the database setup, I used MySQL in both portions of the process, both Mac and Linux.

I think the most frustrating aspect of communicating during my capstone project was attempting

to set the pieces in motion at the beginning. I could not seem to ask the right questions, no matter how many times I tried. Learning to read between the lines of the capstone explanation was a very valuable experience. Figuring out what was expected while reading about the capstone process was a process I will never forget. The way it worked out for me turned most of communicating during capstone into an exercise of my listening abilities.

With regards to the computer science courses and topics that I used, the most influential were CS 315 - Internet Programming and some aspects of CS 370 - Software Engineering. Understanding MVC interactions was very beneficial throughout the entire development cycle, as it helped me keep program logic straight. The other topic I was able to more fully understand was the notion of black box testing, as the testing I conducted was fairly exclusively black box in nature. Going along with my previous mention of listening as a big part of my capstone experience, one idea I used that was not from computer science at all was the psychological concept of reading people. It served me well in the capstone initialization phase. Beyond that one theory though, I cannot say I used many topics outside of computer science.

One thing of whose importance I was reminded was to always be certain project data is backed up. I always knew this to be a concern, but the moment my personal computer decided to take some time off emphasized what a waste of time it could be to not back up important data. I can also say that I learned a great deal more about how file permissions interact with a web server. I now know how challenging it can be to view or alter files created by the operating system when one does not have root access to the file system. While it is possible to simply ignore the unwanted file or alter the naming convention within the program so the system looks to another file, one can also temporarily rewrite part of a working server script to delete the files when the web server runs the script. That is not something I would have considered beforehand. One final lesson I took from my capstone was how to initialize pop-up windows in JavaScript with embedded button functionality. This is another place in which I used the knowledge gained in creating the earlier JS buttons on attaching functionality to detached elements.

The biggest problem I faced was probably consistently working on my project after my personal computer started failing. Understanding that sand was available as a backup strategy saved a great deal of time, and using sand allowed me to switch workspaces, relatively seamlessly, and

begin development shortly after the issue arose.

In hindsight, I think the tutor-scheduling system could have been written more quickly using AngularJS, or even simply some jQuery, but I know little to nothing about either.

When I began the project, I did not think it would be quite as web development as there was. I saw the project as more about finding an algorithm that successfully placed tutors into a schedule within a given margin of error. It turns out that, in addition to a sizable web interface, falling within that margin of error was more challenging than expected. As a result, the best the program can do is give the percentage of tutors who are assigned to the maximum number of hours for which they are cleared.