CS 257.00 Software Design
Amy Czismar-Dalal

A Reflection on the Software Design Experience

In the course of learning about the Agile framework in order to respond easily and effectively to a changing environment, understanding how to properly use the variety of tools we have available (resolving merge conflicts, implementing SQL queries in Python and integrating our queries in HTML using Flask) and understanding the perspective of the user (Krug's five questions, analyzing accessibility and performing unit tests on the boundaries of a query's domain), I discovered that the greatest thing I have learned is the importance of communication.

Model of Collaboration.

Our model is distributive; Hashir would upload datasets to GitHub during the in-class labs, I did the main page mock-up, Hashir did the two sample queries and Tony did the About Data page. Because I am the principle architect of UI design, I have had the greatest impact on our website's final appearance. Based on Hashir's sample code we all made substantial contributions to unit testing. We did this by emphasizing collaborative responses to change instead of following a single plan, implementing working software without too much comprehensive documentation, keeping things flexible with respect to each of our individual assignments and making sure that our best work was done when we were working together and not just measuring progress by the amount of lines of code we wrote.

For example, we originally wanted to use the Google Maps API in order to generate an interactive map displaying restaurant locations. Upon realizing that our users don't need to visualize distance to a restaurant), we postponed this feature. And, our original search box (which resembled Google's) detracted from the focus on user stories and the purpose of our website; we shortened it and sharpened its corners to make it less conspicuous. Throughout the project, we balanced aesthetic appeal with an eye to our website's goal.

Software development requires and always has required effective communication; one thing I learned is the value of identifying users with different categories – consumers knowing what's available, restaurant owners inquiring into other restaurants and researchers/food critics seeing what's already been said – so that they can answer Krug's fundamental question, *why am I here and not somewhere else?*

The Development Process.

I set the framework for the project proposal and the first main page draft by providing the first sketches. Then we divided tasks among ourselves and sent our changes over Slack. I initially formatted the background whitespace as an image of a

restaurant, superimposed by opaque text boxes. We kept this original idea and it is something upon which we have continued to elaborate. We prefer to work together in order to avoid having to resolve merge conflicts and opt for consistency throughout all of our HTML/CSS files. Furthermore we've established a style guide – breaking up long lines, using tabs and Camel case while using best naming practices – and this has helped us make sure that we create organized, readable code which isn't overly complex, fulfilling the Agile tenet that working software is more important than comprehensive documentation is.

So, after converting our four JSON files to a single .csv and implementing our database query signatures, we had to revise our metadata in light of what we could not reasonably do. We could search restaurants by name and minimum stars, get attributes (address, average star rating, location and top five reviews) given a restaurant name, look at nearby restaurants and find the average stars for a specified area to assess its general quality; however, we could not consistently retrieve the hours for certain restaurants and maintain many unimplemented queries as of yet; these are *getTop5Reviews*, *getStarRating*, *getRestaurantTimings*, *filterByUtility* and *runQuery*.

Now that we have fully completed the Flask integration, we will be able to add these features in addition to revising user stories to the scope of our project, allowing search results to reflect one of three main user types, including the Google Maps API in JavaScript and updating our metadata.

Key Technical Lessons.

There are a variety of small details – specifying "localhost" as a parameter to the main method, maintaining one primary CSS style sheet (as well as individualized CSS files for each webpage) and judicious use of unit tests – which detract from the main technical lesson which is to correct problems early as they appear. This requires that developers make sure that everyone knows which files are the most recent and push to GitHub often. Specifically, we initially used a single formatting.css page which we later split into three CSS files which share the same repeating code blocks. I wrote many unit tests which failed (they are not supposed to fail!) and so have gained a better understanding.

Key Managerial Lessons.

Finally, but not least importantly, I want to reiterate that developers need to be on the same page, engage in early planning for which data should be used and understand what everyone is doing. The various things which happened (different style sheets, needing to put static and templates into the root directory when integrating our website with Flask and some confusion about the use of a main function for our API) indicate that when managing a team of developers, they must not be doing different tasks at once without direction. The use of starter code and the instructions provided in class were critical in terms of setting the foundation for our success. Google style docstrings, learning about SSH and SQL and checking our comprehension of the

readings (for example, whether an API needs a main() function) were our framework for efficacy, and in retrospect to the *Previous Code Revision* assignment, it's important to keep track of our changes whether we track them through judicious commits, our Slack channel or through clarifying doc-style comments. Organization is the key.