Matthew Stucki
CS 4986: Computing Internship
3rd April, 2014

Technologies and Takeaways: Accomplishments in Interning

In my last 5 months working for Crawford Technology Services (CTS) as a software engineer / consultant, I have been a part of several different projects of different scale; a few large-scale and several small-scale. These projects spanned several different technologies and platforms (mostly web-based) including MVC4, Drupal, Moodle, and .NET (desktop application). However, beyond the platforms and technologies I learned and used, I was able to gain real-world insight and knowledge that has helped me grow as a developer and as a marketable asset to the technology industry. I have described in detail each of the platforms, technologies, and takeaways below.

Technologies Utilized:

MVC4:

The large-scale projects I worked on for CTS were based on the MVC4 (Razor) platform that is part of the .NET Framework. The projects I worked the most on were two HIV-care websites from GlaxoSmithKline (GSK), www.hivpractice.com and www.myhivhangup.com. Both were based on MVC4 as it is a more mobile-friendly platform that harnesses the incredible power of the .NET Framework. Hivpractice.com is a provider-care website for doctors to request visits from their pharmaceutical representatives and build patient-care packets for newly-diagnosed patients. Myhivhangup.com is a website for newly-diagnosed patients with 'hang-ups' towards treatment that aims to alleviate the fears of new patients.

Hivpractice.com was the easier website to develop as it did not require a large amount of JavaScript development and the 'patient-packet-builder' used a PDF library that handled the majority of the work for building the PDF-based packets. However, the patient-packet-builder is a neat feature, as it allows users to drag-and-drop packets from a slide-bar on the left into a 'working' document and re-arrange the page numbers to make a user-specific packet out of 30 pages of PDF documents. Once the 'packet' was built, the user could save it to their computer or directly print it from the site. The request-a-representative feature was relatively easy to integrate as it was a form with fields/validation and once successfully complete would generate e-mails to the appropriate representatives with calendar invites. The site also had a neat 'polls' feature that was entirely data-driven and would poll individuals with a new poll every month that could be specifically targeted to them based on the type of clinic they were representing. This was set up entirely in different database tables that checked to see which type of clinic the user was representing, then grabbed the associated polls and populated the polls on each page as they were loaded (based on date). The only drawback was that this data had to be manually entered into the database regularly (monthly) to keep new polls available to users.

The more interesting, complicated website was myhivhangup.com. Myhivhangup.com was simple in its use: allowing users to register for updates to follow their (supposed) progress through treatment and help ease their concerns while the process moves forward. However, the website was to be extremely graphics-oriented, and required a lot of JavaScript manipulation to allow for the complex sliding features that were requested. The majority of the development was in JavaScript, creating libraries to control the motion of the 'characters' on the home page and allowing for the 'characters' to be scrolled through (or skipped entirely) as there were no pre-defined libraries for this sort of manipulation. The website also required quite a bit of REST/WCF service development, as GSK requested the ability to register for updates from third-party sites without being channeled to the myhivhangup.com site.

Both of the above websites however, were easy to develop in that MVC4 is an extremely powerful platform. Not only was it easier to define master pages and inheritance amongst pages, even basic tasks like creating and populating textboxes were much easier because of how MVC4 uses data dictionaries to retain and use variables. Also, MVC4 allows for the calling of behind-the-scenes code directly from the UI through Action calls (using ActionResult as the return type) that could either complete a task or return an object (such as a page result, an object (video, etc.), or otherwise).

Drupal:

Drupal development has been my least favorite aspect of my internship. Drupal is a content management system (CMS) based on PHP/MySQL. It utilizes hooks (callbacks) to allow developers to create 'modules' that can be integrated into the default Drupal installation and modify its functionality. The modules are in themselves their own, small-scale application, having the ability to render and return their data to the Drupal UI using their own CSS and formatting. Drupal then simply displays the data where appropriate based on the instructions in the hooks. I have only worked on one Drupal site thus far, www.slabdreamlab.net, and it was for a small, family-owned business. The website is a marketplace for Lego 'slabs' that children (or adults) could use as the base of their Lego landscapes. Most of the work involved in the site was set-up, installation, and modification of the default Drupal Commerce install. The client had a weird layout and UI theme request, and most of the work went into the back-and-forth changes to theme.

Moodle:

Moodle was my more enjoyable PHP/MySQL experience as an intern. Moodle is also a CMS, only it is a more specialized version known as a Learning Management System (LMS). Moodle works similarly to Drupal in that it uses modules to alter the functionality of a default Moodle installation. I have not had the pleasure of doing a full Moodle site yet, so my knowledge is much more limited – I have only developed a role-based user access control (RBAC) module for a pre-existing, well-developed site. The RBAC I developed was for Tropical Smoothies' Employee Training site, of which I only had access to the development site, http://tsc.dev.novologic.net/moodle/login/index.php. The RBAC was used to separate the responsibilities of employees, shift managers, and store managers. The module itself was contained in a Block (a small widget that can be shifted around the site at the users' preference) that was only visible to store managers (the first part of the RBAC). Users with visible access to the block were given a link to "Manage Employees" as well as a drop-down list of current (active) employees, separated by role (Employee, Shift Manager, Store Manager). If the user navigated to "Manage Users",

it was given a list of their current employees in a table, and a list of non-current employees in a separate table. The user could then suspend (fire), reinstate (rehire), delete, or edit any user. This block was important because prior to this, due to the usage of Moodle's Cohorts function (grouping of like users), store managers were currently 'Administrators' that could modify all other employees at all other stores, rather than just their specific store. The RBAC I developed removed that functionality.

.NET:

The purely .NET (C#) desktop application I developed was for a private University that every year created around 100 different brochures for prospective students regarding their programs and how each would benefit the student in their career field. Before the application, each year the University manually updated each brochure with image replacement and text replacement. Each brochure was about ~20 pages in length, so this process was extremely time consuming. The University contracted with CTS to develop an application that would take textual and image input and generate several different files to automate this process; which became a decent-sized project I completed entirely on my own.

To complete this task, I did two things. I first created a base brochure (they all used the same page layout and color scheme) using Adobe InDesign and set up textual and image import areas throughout the document and labeled each appropriately. Then, I developed the application using this brochures labeling system in mind that would generate three separate files, an XML file for storage of the program information and settings for each, a CSV file containing textual information (or file locations) attributed to a variable name that matched the associated brochure, and a JavaScript file that used the settings saved in the XML file to appropriately move the necessary images throughout the InDesign document or re-size current images to their appropriate sizes (for charts, etc.)

This setup allowed the University to be able to store the information for each program, and switch between programs using a dropdown list. The user would select a program, and the most recently stored information would be populated into all of the fields (there were 7 tabs and around 150 fields). The user could then change what was necessary, switch images, etc. and then save the information and generate the necessary files. File generation created two copies of the files, one at the location specified for use by the user, the other in the applications files directory to be used next time the application was loaded. To update the brochure, the user would open Adobe InDesign and simply import the files, starting with the XML, then the CSV, and last, the JavaScript. By doing this, all variables would be imported, then all data, then all necessary modifications would occur and the brochure would be complete. This project took around two months to complete successfully and fully-test, and has been used to generate all of their brochures for the semester of Fall 2014 successfully.

Career-Building Skills & Takeaways:

Patience Building:

One of the main takeaways I have from my internship is patience. As I have been required to handle clients on my own as a consultant (especially for the slabdreamlab.net project), I have learned to be very patient with people and understanding of the fact that in general, people do not know what they want in a computer application. Then, once they have an idea of what they want, they don't understand the technical aspects of developing and testing an application that suits their needs, or the amount of time, energy, and money required to create a robust, large-scale application or website. So, I have become very good at being repetitive (without sounding irritated or condescending) and building a personal relationship with each client I have had to deal with. I have learned that people, as a whole, are more understanding and willing to cooperate when they are dealing with someone they find personable than someone they find overly professional. This knowledge definitely helped me in getting my most recent position as a .NET engineer because it was a large talking point during my interviews.

Non-Technical Conceptualization and Explanation:

Following up on the patience-is-a-virtue takeaway, I also learned how to conceptualize and explain technical ideas in a very non-technical, low-level way. Most of the clients I dealt with directly were not technical individuals who had zero understanding of a website or programming. Being the case, explaining how Drupal or a website as a whole works was very cumbersome at first due to my inability to connect to them on a level of understanding. However, after working with and building a relationship with each client and understanding what fields they were knowledgeable in, I was more able to explain CS concepts to them in a manner in which they would understand.

Workplace Agility:

The largest take-away I had from this internship/position was increasing my overall agility. Every day was a new project, or a new use-case on a project we had completed weeks/months before. On a regular basis we would also have red-alert level issues with a project in production (such as hivpractice.com) that would require us to stop working on current projects and shift gears to the red alert. At first, this work pace was extremely stressful and I felt like I was doing something wrong; but I have realized that in general, in development, a lot of your work will be in maintaining or fixing bugs on projects you thought were complete, and this means months and months after you last worked on them. This requires agility in your ability to remember the project, the language, the platform, and to be able to switch between workloads with ease.

Overall, I feel my experience at CTS was entirely positive. I had a great support base and was able to experience a lot of different situations and work with several different technologies that helped build several of my skillsets that I learned at UWG. I am excited to now be starting a job that I feel is more career-oriented as a .NET engineer at an international company where I feel there is room for vertical movement. That being said, I also feel my experience at CTS taught me usable traits for graduate school if I am accepted, including building my patience and agility.