SwiftAid.org

Tools and Usability Testing Report

(Deliverable F)

Technologies Used

SwiftAid.org was developed using the following technologies:

- Microsoft Access – Database
- ASP.NET/C# - Code
- Visual Studio - Editor

The original intentions were to develop the database in SQL/Server. The code to load the tables was originally completed to load the SQL/Server database in October. Connections to a SQL/Server database were not guaranteed, so we switched to Microsoft Access. Tim Elston had experience with C# and did not know Visual Basic, while Joel Calderone and Yassen AbdEl-Baki knew Visual Basic and not C#. We decided to write the system in C#. While there were plenty of editors available for free, the developers were able to get the Microsoft Development Environment loaded onto their desktops. For consistency, we used the same editor, Visual Studio.

Encountered Problems/Solutions

The most significant technical problems involved getting the server account established and establishing permission for access, both to the server itself and to the database folder. The account was not established until week seven. Then we lost several days because we determined that lab support had not set permissions for us to access the database, but we could not get responses from lab support to set them. After the account was set up and permissions were established, Tim had difficulties with the Windows and IE FTP connections. Files and databases would become corrupted when he attempted to download or upload files. Tim switched to CuteFTP, and he was then able to work with files but he still could not work with the database. If Tim downloaded or uploaded the SwiftAid.mdb for any reason with any tool, the database became corrupted. As a solution, Joel and Tim decided to appoint Joel as the DBA. Tim developed his pages off of a separate development database and then forwarded tables or change instructions to Joel, which Joel then implemented in SwiftAid.mdb. This worked out well. Joel was around most of the time and most changes didn't take that long. Here and there, the one-person DBA slowed our development down.

The second biggest problem was that only Tim could code in C#. With the amount of screens that needed to be coded, a one-man coding show wasn't going to work. As a solution, the team decided that Harry Saito would handle the format and look of the whole system. All Cascading Style Sheet changes would go through Harry Saito. Again, this approach worked out fine. The team stands behind the look and feel of SwiftAid.org. Time was again chewed up with one person working the Cascading Style Sheet. Joel started learning C# the first week of October. While in the end this worked out alright, early on it took Joel away from some of the paper work. That left Tim overloaded with paper work.

Harry experienced difficulties accommodating both Internet Explorer and FireFox. He decided to design features that would work in both browsers. This worked out well.

In the end, Tim completed most of the Provide Aid, Donate, and Emergency News sections, Joel completed most of the Request Aid and Administration sections, and Harry completed the CSS work.

Usability Testing

Because Harry did most of the design in CSS, page uploads are fast. Some formatting had to be accomplished with HTML tables, however, which slows down page loads somewhat, but not significantly. The photos in the Emergency News section were saved in JPG format to speed page loads in those sections. Pages that had no dynamic content such as the Home page and all pages in the About Us section were kept in HTML format rather than ASP.NET so as to optimize their download speed.

The site retains areas of weakness regarding usability. The Transportation registration and Aid Request pages, along with several pages in the Administration remain under construction. This was due to being shorthanded. Joel coded his pages optimally with validation constraints, whereas Tim omitted most validation coding in order to maximize breadth of coverage for prototype display. This left Tim's forms susceptible to data entry error. Were the site to be developed further, this would be an obvious area for improvement, so much the more because it also directly impacts security.

The Credit Card payment form was not scripted because setting up a functional payment system that is linked to an external payment system was beyond the scope of the course. The PayPal payment function was coded to the PayPal sandbox site, which is a fictitious development site that replicates all aspects PayPal payments without connecting to a real payment account. For this payment option to work, the browser must be logged into the SwiftAid.org developer's account.

We feel that the structure of the site is well developed, from a usability standpoint. The About Us, Request Aid, Provide Aid, Donate, Emergency News, and Administration sections are well named, and each section has functional integrity. The registration processes take the registrant through a succession of easily navigated steps. Were the site to be develop fully, we would implement cookies for leaving and returning during the registration process and for logging in and out as registered victims, providers, or administrative personnel.