Jacob Cole, Michael Pittard, Parker Cantu

Dr. Do

9/6/2015

CSCE 4444 - Software Engineering

**UNTextbook Finder**

**Project Description**

As of right now, there aren’t any price comparison websites for UNT students. Our group is going to create a website that will provide a way for UNT students to compare textbook prices against various websites/stores. This will hopefully bring a more UNT-focused experience for comparing textbook prices. The idea is to make it easier for UNT students to find out who has the books they need and, perhaps more importantly, where they can get the books they need at the lowest price.

The environment to develop will require a few tools. First is XAMPP, which will run our website locally in order for us to develop. Next is the Scrapy framework, which will be used to create scripts for web scraping. This will provide us with a structured, object-oriented way of scraping web pages, and allow us to use common functions more than once, thus saving time in our development process.

The website will scrape data from the Voertman’s, UNT Bookstore, Amazon, and Chegg websites using Python scripts. The scripts will store the information of books as JSON, and we can use a PHP call to extract data from the JSON file. The users will use our interface we create to search and compare the prices from those websites. Our interface will be created using HTML, CSS, and Javascript with jQuery.

There will be multiple pages to our website. The first is the home page with basic information, including what the site is and how it works. The next will be the search page that actually serves as the place to compare textbooks. Once the user inputs a search, the page will update using AJAX with the information that our scripts come up with.The last page will simply be an “about us” page, which will include our three names and why we made the web page.

**Project Timeline**

This project will take a total of 8 week to develop, test, revise, and deliver to the end user (starting 9/18/15.) Project planning has already been conducted. Interface software and the web scraping software will be developed in parallel and take 2 weeks. However, the controller must be developed after those two are written, taking another 2 weeks. Once the website is finished being modeled, testing and revising will be conducted for 4 weeks. A manual will be written in parallel with the project which will take 8 weeks to finish. A step-by-step process, a Gantt chart, and PERT diagram are listed below. These will be revisited at a later date to ensure correctness and to make adjustments if needed.

   1.1  Project planning

   1.2  Manual

(8 Weeks)

   2.1  Develop interface

(2 Weeks)

   2.2  Create web scraping scripts (can be done at same time as number 2)

(2 Weeks)

   3.1  Build controller that connects the interface and backend (scraped data)

(2 Weeks)

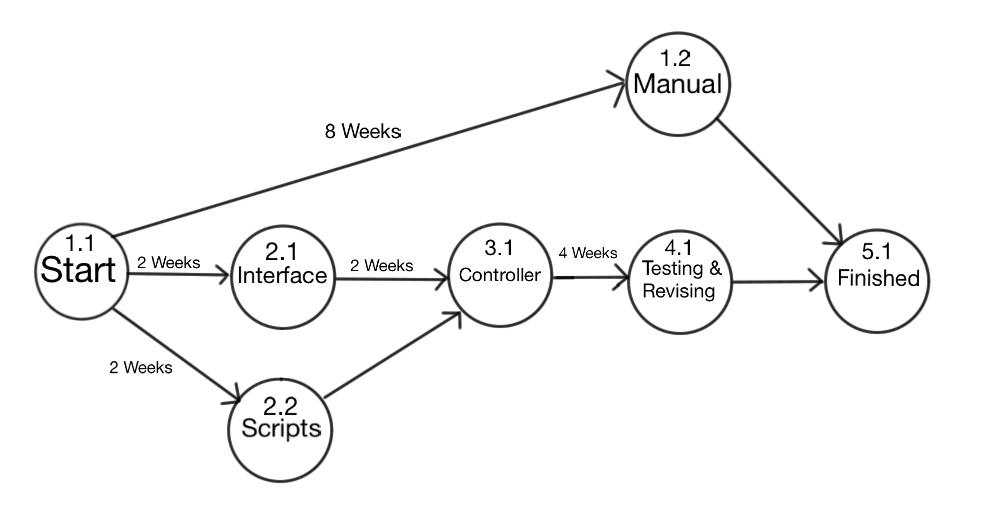
   4.1  Testing & revising

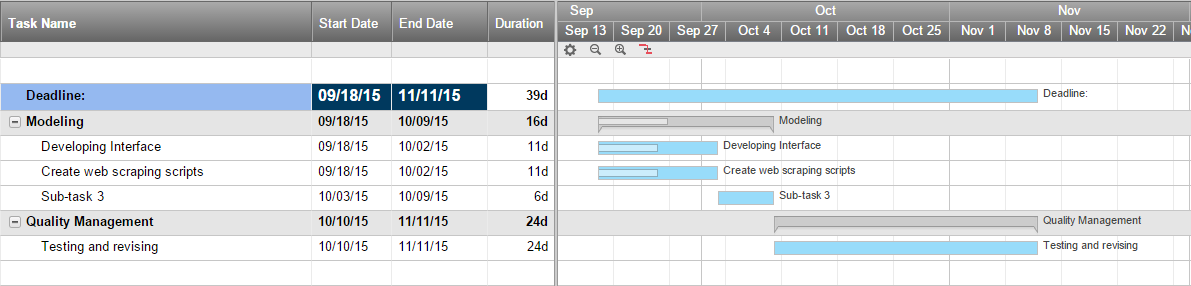
(4 Weeks)

   5.1  Deployment

(Total time: 8 Weeks)

**PERT Diagram**

****

****

**Risk Analysis**

Top 5 Risks

1. Connection timeouts/third party websites down
2. third party web page format change
3. Flawed logic
4. Inaccurate dependencies
5. Time spent on design decisions

Risk Management

*Connection timeouts/third party websites down*

Scraping and using APIs to create our website means that if a third party website goes down, our web site will be directly affected. This risk should be monitored by testing the application weekly. In the event that a third party website can’t be reached, the most recent information from that website will be pulled from a JSON file. An error message listing the websites that are down will be shown to the user.

*Third party web page format change*

Another risk of scraping and using APIs to create our web site is that a lot of our web site’s stability will rely on third party web sites’ format or API logic remaining consistent. Third party changes can range from being harmless to our web site to absolutely breaking a major function of the website. Format changes should be monitored weekly. In the event that a third party’s web page format is changed, appropriate changes will be made to the scripting logic of the application.

*Flawed logic*

Sometimes ideas and logic don’t end up mixing well in the early development processes. Bad communication can also lead to skewed or downright broken logic. The logic used by the application should be revisited periodically by the team to make sure that it still makes sense. In the event that flawed logic is discovered, either through testing or evaluation, action should be taken as soon as possible to correct the logic and return the application to a working condition.

*Inaccurate dependencies*

Many components of the web site one person will be working on will be heavily dependent on another part of the web site someone else is working on, and vice verse. Dependencies should be monitored when making small changes to the design of the application to ensure that any changes in dependencies are noted by the team. In the event that a dependency is changed, action should be taken to redesign the flow of the application to support the new dependency.

*Time spent on design decisions*

It’s easy to get caught up in the fine details of a project, even when you’re not to the point of implementing the idea yet. Design decisions, whether they be crucial to the development of the project or purely aesthetic, should be made in a timely fashion so that the team doesn’t get caught up in design choices when important progress could be made. In the event that the team is caught spending too much time on a single design decision, a decision should be made at that time so that the development of the project may continue.

**Project Report**

*Team*

        Jacob Cole (leader)

                    - backend scripts

        Parker Cantu

                    - interface

        Michael Pittard

                - controller (mostly depends on backend/interface being done)

- help interface/backend while waiting for them to finish up

*Repository policies*

* Using GitHub as our remote repository
* We shouldn’t have conflicts because each of us will ideally work on separate components (backend, interface, controller)
* If a conflict arises, use Facebook Messenger to communicate the resolution
* We will each have our own branch from master with our name being the name of the branch
* We will merge our independent branches into master once our features are ready

*Progress*

We have our repository set up with GitHub, and have practiced with both the GUI interface and the command line. Jacob has the most experience with GitHub and wrote up a document explaining the basics of pulling, committing, and other common functions that we will be using to work on our project.

We have come up with the foundation for our logic and how the backend system is going to work. Also, we have successfully installed the Scrapy framework and tested that it pulls data from a web page. While each team member has been delegated specific areas of the application to work on, the nature of the dependencies in our program means that strictly adhering to the delegations would slow the development process down.

Because of this, we’ve decided that if someone is waiting to work on a dependent function of the application, then they will help another team member work on their part of the project until their dependent function can be worked on. This is at the discretion of the team, especially if the function or feature is small and will be finished soon, since it won’t always make sense to have two people work on something. Having the three of us collaborate this way allows us to maximize our time spent on the project in terms of efficiency.