A Registration Planning Utility for the University of Pennsylvania

Amanda Leicht – aleicht@seas.upenn.edu
Matthew Moore – moorem2@seas.upenn.edu

Faculty Advisor: Susan B. Davidson
1. Abstract

For over a decade, the “ultra-modern” Penn-In-Touch was the only utility available to students for their bi-annual course registration. During this time, registration required not only considerable time and effort, but the knowledge of and ability to consult five to six separate resources simultaneously. The process of creating a working schedule involved the Penn-In-Touch registration window, the Penn-In-Touch course planning guide, the Course Description Registrar, the Course Timetable Registrar, the Penn Course Review website, and a separate spreadsheet to consolidate all the aforementioned information and avoid time conflicts. Simple tasks ranging from locating other classes your favorite professor is teaching to selecting an Arts and Letters requirement which meets Monday/Wednesday/Friday at 10:00am were nearly overwhelming for the average student—there was simply too much information to sort through manually. It was these problems that CourseBook, our course planning utility, sought to solve.

CourseBook, at its inception, aimed to consolidate this information into a system developed with the Google Web Toolkit which was easy to use, search, and explore. The system provided diverse functionality, including the selection of courses based on specific criteria such as time constraints, professor, difficulty, and requirements satisfied. It helped with creating and storing course schedules, matching courses to graduation requirements, and viewing schedule summary statistics. All relevant course information was compacted into an elegant and easily accessible display, consolidating pages of information for easy browsing. CourseBook was to provide the tools to students to save countless hours and frustrations, make more informed course selection decisions, and generally improve the quality of the course registration experience at Penn.

Unexpectedly, however, the University released a new “Course Search & Schedule Planning” tool late in the development of CourseBook. The utility provided much of the intended functionality of CourseBook, but was a mere shadow of our vision for CourseBook in many respects, such as its failure to integrate Penn Course Review information. CourseBook, in response, took on a new challenge: to provide an easier to use, more student friendly interface than the University utility’s, while still providing the originally intended functionality. With this new goal established, and with student feedback, a valuable resource the University often fails to consider, CourseBook provides a clean, intuitive, and visually pleasing interface for course registration—and a challenge to the ideas behind the University’s new system.
2. Related Work

2.1 Related Works

Penn Student Projects

Course review information was first made available online in 2002, with the creation of the Penn Course Review. It was a senior design project by Howie Vegter and Steve MacCrory which aimed to digitize the course review information that had been distributed on paper since the 1950’s (MacCrory & Vegter, 2002). Vegter and MacCrory managed to form a partnership with the Student Government and Undergraduate Assembly who agreed to support their project. The project was initially deemed a huge success, as it provided a much needed upgrade to the nearly-unusable solution the University had originally created. However, over time it has become a burden as the University continuously tries to find people to maintain and support the site. Vegter and MacCrory’s project gave us some hope for obtaining University approval, but in the end, its non-sustainability may have actually hurt us, as the University seems unwilling to support future senior projects. In addition, we have used the current incarnation of their website to obtain all of the Penn Course Review data we have used in our project.

One of the first attempts to make a searchable Course Registrar at the University was a 2004 senior design project called Registroogle. Andre DeClercq and Shreyans Bhansali, the creators of Registroogle, set out to build a system where you could search for classes, plan your schedule, and ‘one-click’ register, instead of going through the lengthy registration process on Penn-In-Touch (Bhansali, 2004-2005). However, the University denied them access to both the course database and the system which registers students for classes. In the end, their project was a web based application that allowed users to “search through all the courses offered at Penn using a combination of over 10 different criteria, read details about the course, and build a schedule that tracks conflicts and statistics for the proposed schedule” (De Clercq & Bhansali, 2004, p. 1). This project demonstrated the strong desire for a better course planning tool at the University. Unfortunately, the system was not implemented in a sustainable way, and disappeared when the creators graduated. It also lacked an aesthetically pleasing design, and focused mainly on functionally over usability. The main lesson we learned from Registroogle was that we should not expect the University to cooperate with our desire to obtain the course registrar databases, and although we made a valiant effort to do so, we were not surprised when they denied granting us access.

The most recent student attempt to build a searchable course database was a web tool called Registrar Search developed by Gayle Laakman in 2005 (Laakman, 2005). Her tool scraped all of the course information from the Penn Registrar and allowed users to search for courses based on time, days, subject, or professor. Although her application lacked in rich functionality, it was another demonstration of the desperation of students for a course searching utility. Unfortunately, Gayle has since graduated, and has moved her site from the www.glaak.com domain name, rendering the site
unavailable. However, we were fortunate enough to get in contact with Gayle and she graciously showed us her (now fairly outdated) parsing utility, which we referenced as a starting point for the core of our parser.

In planning CourseBook we took to heart a lesson from a recent student project, ClassBuster. ClassBuster was a tool developed by University of Pennsylvanian student Danny Panzer that allowed users to automatically check for openings in full classes and send an email or text message when a class became available (Panzer, 2006). However, the University explicitly warned students against the use of this program, as the application was not authorized to access Penn-In-Touch, and it was eventually taken offline (Yahalom, Registrar Busts ClassBuster, 2006). Although our system will not be able to provide this functionality for the same University mandated reasons, it provided us with some important lessons about dealing with University policy. It is for this reason that we have decided to require registration with a @upenn.edu email address to access our application.

Official University of Pennsylvania WebPages

Prior to March 2008, the University Registrar was the only University supported source of course information. One page provides course descriptions and prerequisites, and another provides course timetables for the semester. There is a link to the Penn Course Review, a site where professor and class evaluations are stored. Registering for classes requires logging into Penn-In-Touch, and manually entering the desired classes (they must be written down or looked up beyond the registration utility). During advanced registration it fails to inform of any conflicting selections, nor provide any visual representation of a requested schedule, apart from a list. It has long been recognized that the Registrar is a poorly designed maze of links and pages that makes course planning a nightmare. In Fall 2007, the Daily Pennsylvanian reported that, while the Undergraduate Assembly understood the incredible need for a new system, the “basic opinion [of Information Systems and Computing and the Registrar] was that they'll replace it, but not for another four years or so (Kaplan, 2007)."

One department at Penn has made course selection slightly easier. The Critical Writing department provides a tool to search writing courses by term, program, department, instructor, meeting day, meeting time, and keyword (The University of Pennsylvania, Critical Writing Courses, 2008). This is an excellent tool for finding writing courses, and again demonstrates the need for easily searchable course information. Unfortunately, since it makes only writing courses available for browsing, it is of little use to the overall student population.

In early March 2008, the University unexpectedly announced a Course Search and Planning tool which was to be released on March 17, 2008. This tool allows students to search the Registrar timetables, as well as view detailed information about classes (description, requirements fulfilled, prerequisites, etc). The tool also allows students to create mock schedules, but registration must still be manually done through Penn-In-Touch. The application does not provide access to any data from Penn Course Review. While the tool does include many of the same features we proposed for CourseBook, the site is both aesthetically lacking, and can be confusing to use. While much of the student feedback has been positive, most are aware there is certainly room for improvement (Bell, Students: Creating a
Mock Schedule Made Easier, 2008). The site seems to have been designed with minimal student input, as much of the search criteria and information displayed is not relevant for most students planning their schedule. With such vast resources, it is disappointing that this is the best solution the University could come up with to solve the glaring problems of course selection.

Non-Penn Related Projects

Other universities faced with the problem of creating tools to aid in course selection have pursued a variety of alternatives. While many have produced propriety systems, the University of Alberta chose to adopt a student project called BearScat (a play on the name of their ‘Penn-in-Touch’ equivalent called BearTracks). Developed by Steve Kirkham at the University of Alberta, BearScat makes it possible for users to search for classes based on subject, and plan a schedule to fit those classes (Kirkham, 2007). It allows students to directly register for these classes, receive notifications when a full class has a space open, and add/drop/swap classes. It also provides a listing of required course texts, and access to professor ratings. Although this application was developed independently by a student, the University has provided access to its databases and has even funded BearScat in recent years. According to the BearScat website, last year the application “served 10.6 million hits, 1.2 million transactions to [the University of Alberta’s registration system], 150,000 notifications, and 135,000 designed schedules” (Kirkham, 2007). This project is a strong example of the increased value potential and feature possibilities for a system that is University supported.

As an alternative to building a system, many schools have opted to purchase a pre-packaged solution. PickAProf.com, which was developed by Chris Chilek and John Cunningham, contracts with universities to provide enhanced functionality to course planning and registration (Chilek & Cunningham, 2007). It began as a tool for students to rate professors, much like Penn Course Review, but has expanded to include schedule and degree planning. Users can browse courses, see professor ratings and grade histories (i.e. how many A’s or F’s a professor has given in past years), and create a timetable with their requested classes. Additionally, users can search for textbooks for their classes. Adopting such a system would have been a huge step in the right direction for Penn, but the University has already stated that it does not plan to outsource Penn Course Review (Yahalom, 2006). Gabe Kopin, the chairman of the Student Committee on Undergraduate Education states that PCR is unique in that it has a "more thorough and diverse representation from the student body that its competitors do not."

The concern is that students that really enjoyed, or really did not enjoy a class would be the only ones motivated to enter data, and the data would not be a representative sample. However, it is apparent that tools like PickAProf are in high demand, as they boast a user base of 180 universities.

2.2 A Closer Look at the University’s Penn-In-Touch Course Search and Planning Tool

In March 2008, the University launched the Penn-In-Touch Course Planning and Scheduling Utility. The system allows students to search the course registrar on 12 different criteria, and view detailed course information, such as the description and prerequisites. Users can add classes to a ‘Course Shopping Cart’, and create mock schedules. While this system is a huge step in the right direction, it is still lacking many features that we consider essential to this type of application.
The most glaring omission from the University’s tool is the ability to search Penn Course Review data. In order to retrieve ratings on classes and professors, students must still separately consult the Penn Course Review website. This adds an unnecessary step to the course selection process, and is a feature that should have been placed at a higher priority. The utility also lacks the ability to register students for the courses they have selected on their mock schedule. Students are still required to manually enter their classes into the Registration page of Penn-In-Touch. While our utility could not have hoped to achieve this functionality, this should have been a key feature for the University’s system.

The other problem with the University’s utility is that it appears to have been designed with very little student input or feedback. While the ability to search by department, professor, course number, requirement filled and description text are useful, many of the other search fields are fairly irrelevant to students searching for courses (See Appendix A). Being able to search by the “first day of the week a class meets on” does not allow students to find classes that do not meet on Fridays, and you also cannot find classes that start exactly at 1:00pm, only those which start at or after 1:00pm. Searching for classes above or below a certain number (i.e. below 200) is almost useless given Penn’s seemingly arbitrary numbering system. Much of the information that is prominently displayed in the search results, such as the maximum enrollment, or the status (which seems to be always ‘open’), while important, could probably be placed in the detailed course information, as it is not vital data.

In many ways, the University’s utility is not aesthetically pleasing. It has a fairly basic ‘web 1.0’ feel and layout and is neither very dynamic nor engaging. The course information returned from search is displayed in a large table with each section listed on separate row. It appears cluttered and at first glance is entirely overwhelming. The mock schedule creation is awkward at best, requiring multiple clicks to add or remove courses from the proposed schedule. The colors are often distracting, and the navigation between pages is non-obvious and confusing.

The tool also fails to display information that is useful to students planning a schedule. When on the mock schedule page, the only way to see any information besides the course number (such as meeting times or professor) is to add the course to the schedule, or click on the title to bring up an intrusive popup window. No statistics are shown about a proposed schedule, such as requirements fulfilled or hours of class. The tool also does not alert you if you are missing a required lecture or recitation.

While it is easy to tear the University’s system apart, we can also learn from what they seem to have done correctly. Unfortunately, much of what they did correctly were things that CourseBook was implementing as standard (and better) anyways, but the University still managed to provide a tool which functioned properly, provided a good deal of search options, and contained a rudimentary schedule builder. In order to compete with this utility, CourseBook had to make sure that everything that the University did right, it would do better.

2.3 The Advantages of CourseBook

The overarching goal of CourseBook is to make the course selection process as simple and easy as possible. To achieve this goal we have continuously sought feedback from students in order to
determine what is most important in searching for courses and planning a schedule (see Appendix A for summary and analysis of feedback).

Our first goal was to allow students to search the Registrar by criteria relevant to schedule creation. Along with the basic searches for department, course number and professor users can search for classes that meet on specific days (i.e. on Mondays and Wednesday but not Fridays), start and end at specific times, and fulfill certain requirements. Most importantly, students can search classes based on historical Penn Course Review data; finding all courses in the current registrar with an average difficulty below 2.0 or with an average course rating above 3.5 is now possible and incredibly easy. Users also have the ability to directly search Penn Course Review ratings, and receive all of the same information found on the PCR website.

Our second goal was to make the user interface as easy to use and understand as possible. We decided to group sections by course, and allow each course heading to ‘collapse’ and hide all section information. This helps in making the section listings much more readable. To keep the listings as uncluttered as possible, we left off ‘extra’ information such as the maximum enrollment, opting instead to put this information in the detailed course information dialog.

The schedule creation page is clearly laid out and easy to understand. Sections that have been added to “MyCourseBook” (a shopping cart) are grouped by course and displayed along with their vital information: meeting time, type (lecture/lab/recitation), and professor. Detailed information is easily obtained by clicking the section number. This allows users to easily see what they are adding to their schedule without being overwhelmed by information. Sections immediately appear on the schedule when clicked, and any conflicts or missing components (i.e. missing a lab required for a lecture) are prominently called out. Finally, relevant statistics for the current schedule are displayed, showing the number of credits, hours of class per week, and requirements filled by the selection of classes.

Although we were not able to implement all of the features we had originally imagined, we feel that we have created an exceptionally useful tool that will meet the needs of most Penn students.
3. Technical Approach

3.1 The Parser

The system which comprises CourseBook is divided into essentially four different (though not disjoint) parts. The first part is the area which deals with extraction of information from the University systems for insertion into the CourseBook databases. As is discussed below, the University denied us access to registrar data directly, and so the data had to be scraped from the website(s), processed, and then inserted—a relatively complicated procedure.

The data collection and processing part of CourseBook’s implementation has been designed as a parsing utility run locally from an administrator’s machine, which collects information from the web, processes it, and inserts it appropriately into the database tables. The parser itself was coded in the C# .NET language, partly for the power provided by the language and partly due to ease of implementation. C# provides a solid framework for both regular expression matching and for standard string format matching and alteration, which was adaptive for the poorly formatted nature of the page source. Added to that was the fact that we had prior experience in pattern matching utilities in C#, and that the outdated parser provided by Gayle Laakman was also in .NET, so the language seemed like a natural choice for a parsing application which need not necessarily exist online.

The workflow of the parser is a several step process, and on the average will run will complete in less than twenty minutes on a University network connection. The parser initializes and the form waits for activation from the user, upon which it dumps all information currently in the databases, and then navigates a WebBrowser control to a Penn Course Review page. Penn Course Review is login protected, so a redirection occurs which requires the user of the utility to log in with their Penn Key. Upon a successful login, the program then extracts the cookie (and authentication) generated from this login for use in Course Review information extraction of web pages which are PennKey login protected. The parser then scrapes the Registrar timetable for a list of department codes (e.g. CIS, MATH), and for each code downloads the appropriate page for each department on the timetable, registrar, and Penn Course Review. The HTML is parsed by the utility, scanning for course numbers, sections, and relevant information through regular expressions and pattern matching (of which there are a considerable number of instances and cases), formatted and organized, and inserted into the appropriate tables in the database.

3.2 The Database

Information stripped from the web is stored in a MySQL database which is hosted on db.stwing.upenn.edu. The schema design of the database was important to keep storage space low and the queries as efficient as possible given the limited capabilities of our server. Seven tables in the database are relevant to the storage of course information, while four deal with handling client-side
user information, such as usernames and passwords. The seven data tables are Courses, Sections, Reviews, ReviewDetails, Meets, Departments, and Satisfies. The four client-user tables are Users, Schedules, CBookCourses, and ScheduledCourses. For additional information including table schemas and descriptions of columns, please see Appendix B.

**ENTITY RELATIONSHIP DIAGRAM – DATABASE**

![Entity Relationship Diagram]

**Figure 1 - Entity Relationship Diagram**

**Section 1 – Course Information Tables**

a) Table: Courses

Contains information relevant to each course listed in the timetable. Used in conjunction with the Sections, Meets, and Reviews tables to extract course information during a search of the Registrar.

b) Table: Sections

Contains information relevant to each section of each course listed in the timetable. The sections reference the CourseID of the course they are within—for instance: if the section is 001 and it references ACCT 101, then the whole section identifier is ACCT 101-001. Used in conjunction with the Courses, Meets, and Reviews tables to extract course information during a search of the Registrar.
c) Table: Reviews
   Contains course information summaries pulled from the Penn Course Review website. Data is limited as it was extrapolated from a simple search of the course by department and course number. Used in conjunction with the Sections, Meets, and Courses tables to extract course information during a search of the Registrar.

d) Table: ReviewDetails
   Contains detailed Penn Course Review information for all available semesters. All data is retrieved about all classes and sections current and past, and is stored here. Used for in-depth searches of Penn Course Review.

e) Table: Departments
   Table mapping course code acronyms to their proper titles. Used for lookup of proper name of a department in display.

f) Table: Meets
   Due to the nature of the time schedule with a section having not only the ability to meet on multiple days (e.g. MW 2-3:30 and F 1-2), but multiple times on the same day (MWF 12-1 and M 4-5), a table was given to map the sections to the times they meet to allow as many mappings as necessary for the section. Used in conjunction with the Sections, Courses, and Reviews tables to extract course information during a search of the Registrar.

g) Table: Satisfies
   A mapping of the sections to the college requirements they satisfy, using information extracted from the time table. Required because of a section's ability to satisfy a multitude of requirements. Used for searches where the requirements satisfied field has been specified.

Section 2 – Client-User Interaction Tables

h) Table: Users
   Contains a list of the emails of all users registered to the site with properly hashed and encoded password and security information. Used for site registration and logging in.

i) Table: Schedules
   Contains information about each of the schedules a user has created. Does not contain information about which courses are on the schedule.

j) Table: CBookCourses
   Contains information about courses that users have selected for adding to their schedules. Does not contain information about which schedules the courses have been added to.

k) Table: ScheduledCourses
A many-many mapping of Courses to Schedules that keeps track of which courses have been added to which schedules. Required because of the ability for users to have many schedules and different or the same courses added to each.

### 3.3 Client-Server Communication and Server-Side Functionality

For developing the web application aspect of CourseBook, we employed the Google Web Toolkit (GWT) ([code.google.com/webtoolkit](http://code.google.com/webtoolkit)). GWT is an open-source Java framework which helps make writing AJAX / Java Script web applications easier by providing a rigid development framework that masks many of the browser specific quirks of JavaScript. Additionally, GWT allowed us to develop the application in Java, a powerful, object-oriented language, far more effectively than would be otherwise possible. GWT translates Java into browser-compliant JavaScript and HTML on compilation, which helped in writing code that is more modular, object intensive, and testable. Instead of focusing the majority of our time on fixing small, browser-specific errors in our JavaScript, we were free to focus more on developing the features of our application.

The GWT framework provides for communication from the front end or client-facing portion of the web application to the back end or server-side portion by use of Remote Procedure Calls (RPCs) to Java Servlets. As such, the web application must be hosted on a server which supports Java Servlets, and one of the main server applications which does so (and our choice for hosting) is Apache Tomcat. GWT provides an asynchronous method framework for making these RPCs, where serializeable objects and primitive data types can be passed with ease between method calls (as parameters or return values) in the application. The asynchronous nature of these method calls required careful attention to race conditions, but the benefits of such a rigid and securely implemented procedure far outweighed having to manually construct secure communications between Java applications and Servlets.

The client-facing portion of the application is restricted in what packages it can import and what types of calls it can make. For instance, the client cannot utilize the Java database package (JDBC). Should the client wish to retrieve data from the database, it must construct an asynchronous call to a server method, which will then use the JDBC package to establish a connection to the MySQL database, retrieve results, process them into a serializeable object, and return them. Any such interaction with the database must be made in this way.

```java
public class Course implements IsSerializable {
    private String dept; // Department
    private String courseNum; // Course Number
    private String title; // Course Title
    private String desc; // Course Description
    private String prereq; // Prerequisites
    private List sections; // A list of Section Objects
    private String extraInfo; // Lecture, Recitation or Group requirements Exist
    private String flag;
}
```

Screen Shot - A Course Object
On the server-side, the methods are constructed to be robust, as fast as possible, and secure. Every time a new client is created, a new instance of the server is invoked and a connection to the database is established. The method required by the client is then executed with the appropriate parameters, and the server uses JDBC to retrieve a ResultSet or simply execute an update. For methods dealing with the retrieval of course or section information, the server must wrap all of the returned rows up into a list of Course objects, because the client cannot understand JDBC objects, nor are they serializable. Each of the Course objects contains a list of Section objects, each of which contains lists of meeting times and review information, and all are serializable as required. The entire process of wrapping up the ResultSet is completed in $O(n)$ time on the Tomcat server (where $n$ is the size of the result set returned), while the database itself handles sorting the ResultSet beforehand, and can be assumed to run in $O(n\log(n))$. Splitting the workload between the database and the Tomcat server is important in helping to reduce the processing load on the application during times of high volume.

For methods that deal with processing user information, the largest concern is in security. Passwords are sent to the server as plaintext, but must be encrypted before being stored. The current standard Java String encryption package (BCrypt) is used to hash the password strings before being stored, and hash comparisons are made when checking logins. CourseBook does not involve monetary or sensitive data transactions which require intensive regulation of login information, but such practices are good to put into practice no matter the size of the application, hence their inclusion.

### 3.4 The Client

The client side application is built using hundreds of ‘widgets’, which are reusable GUI components. The most common widgets are Panels, containers that hold other widgets and provide an easy way to layout the elements of a page. Other Widgets include TextBoxes, Buttons, and Tabs. (For a full list of built-in UI widgets, visit http://code.google.com/webtoolkit/documentation/com.google.gwt.doc.DeveloperGuide.UserInterface.html). Developers can also define their own custom widgets by extending the widget super class “Composite.” This is how a large number of the components of our page were built.

Search Interface
The search page is comprised of two main TabPanels; the Registrar Search and the Penn Course Review Search. Each of the panels displays the search fields pertaining to that particular search, which were determined using feedback from students on what criteria is most important. The Department box is pre-populated so that users do not have to know department code abbreviations and can browse through the numerous departments at the University.

A user is able to enter their search criteria into the form, using as many or as few of the options as they wish. Upon hitting “Search”, the information in the form is examined and built into a MySQL query, which is sent to the server side. The server returns the resulting list of serialized objects, which are easily unwrapped into the necessary Course, Section or Review objects.

All search results are displayed in a new Tab below the search panel. We felt that adding tabs of results was far more user friendly than replacing the previous search. This allows users to have multiple Registrar and Penn Course Review searches open for easy reference, and eliminates the need to re-do a search.

![CourseBook Registrar Search Form](image)

For the Registrar search, the unwrapped Courses and Sections are displays in a Grid widget with each section receiving its own row. The sections are grouped by Course, and each course is ‘collapsible’. Collapsing a course hides all of the section information, a feature we think is essential to keeping the page uncluttered and readable. The vital information (the section number, professor, meeting times, and Penn Course Review ratings), for each section are displayed in the grid, while further detailed information is available by clicking on the section number. The detailed information (description, prerequisites) is displayed in a Window panel, which overlays on the grid. This window can be dragged to different places on the page, and was, in our opinion, the least obtrusive way to display the ‘laundry list’ of detailed information.
Users can add a course to their “My CourseBook” page by clicking the associated Add button for the section. This triggers the client to package all of the section information into a CBookEntry object, pass it to the server to be stored, and pass it to the MyCBPanel (the My CourseBook Page) to be displayed visually. The sections a user has added to their page are stored in both the database and on the client side in order to reduce the number of server side calls our application must make.

For the Penn Course Review search, the information retrieved is displayed on the tab in collapsible Disclosure Panels. Each panel contains all of the Penn Course Review data for a class, and allows the user to click on the course title or the professor name and launch a registrar search for those criteria. We felt this was an excellent way to connect the PCR search back to the registrar search, as it integrates components of registrar search, but provides all the functionality of Penn Course Review. Due to the enormous quantity of data retrieved for each course in this method, the initial search returns all information, but unprocessed. Processing of the information takes place when the user activates the specific course entry. This design decision was chosen to keep the length of time between activating the search and when the results are retrieved as minimal as possible. Information displayed in the result includes summary statistics of ratings, professors who have taught the course, and semester by semester detailed rankings of course criteria, all drawn from the ReviewDetails table.
CourseBook requires that all users be registered in order to use the utility. While originally we were going to allow both registered and anonymous users, we soon found a number of reasons to disallow anonymous users. We could have allowed users to search the registrar and build temporary mock schedules without logging in, but the number of ‘special cases’ we had to account for when a user went from being anonymous to logged in were staggering. Therefore, we decided that the registration process was simple enough that registration would not be much of an annoyance and that it would provide an overall better and more consistent experience for users.

To register for CourseBook, a user only needs to fill out a short form that asks for a valid @upenn.edu email address, a password, and a security question and answer to aid in retrieving a forgotten password. After this information is entered, the client passes the data to the server where the email is inserted into the database. This triggers the system to send a confirmation message to the user’s email, to ensure the user owns the email address they entered. The email contains a confirmation code, which is then entered into the site. After the address is confirmed, the user is logged in, and redirected to the search page.

My CourseBook

After searching for courses, a user needs to be able to add their selected sections to a schedule panel. On the CourseBook panel, courses are initially received as CBookEntry objects from the search panel, which are then loaded into CBBox disclosure panels grouped and sorted by the department and course number. Courses that were added in previous sessions are retrieved from the database tables at login, via a call to the server, and returned populated with all relevant information as if it came through from the search panel. Courses are then displayed on the right side of the screen, and a randomly chosen color from a preselected set of colors is applied to it, if it had not been previously assigned a color in a prior session.
CBBox panels themselves contain DisplayRow panels, each of which contains a checkbox for adding the course listing onto the schedule. When a checkbox is checked, the CBookEntry, the row, and the color are passed up to the main panel and then over into the currently active schedule panel on the left side of the window. The schedule panel, which is comprised of 37 rows of 6 blank panels, calculates which blank panels to remove to make way for new panels representing the courses, whether or not the schedule is currently displaying a sufficient range of times to properly view this addition, and whether or not this addition creates a conflict with a currently existing course. In addition, the schedule panel provides click functionality for each of the panels, automatically resizes itself to fit new additions, and recalculates relevant summary information about the schedule.

Figure 5 - My CourseBook with schedules on the left, and courses on the right

The summary information is dynamically updated below the schedule panel as additions and removals of courses are performed. The summary information informs the user of which courses are in the current schedule (some may not have meeting times), how many hours of class, the credit load, the average difficulty and ratings, and requirements filled by the schedule. It will also display information
regarding conflicts on the schedule, and whether or not the current registration fails to meet caveats such as a missing registration for a recitation.

The data processing for these calculations is entirely performed on the client side, with the only server interaction being updating the schedule in the database as courses are added and removed. Conflict calculations take place in the schedule panel itself, but the panels for the courses contain all information and handlers relevant to keeping track of their status. As an added benefit, the object oriented design allows for keeping the schedule calculations separate from storing information about each of the courses. The schedule panel need only remember what panels it contains. Because a schedule itself is simply an object, the user can at any point create a new schedule and switch between any number of schedules. While this requires a minor amount of loading time as data is pulled back and forth from the database and properly constructed, this functionality is valuable in letting the user store multiple configurations of courses and decide between them at a later point.

3.5 Principle Technical Challenges

Each of the four systems detailed above provided us with interesting and unique technical hurdles to overcome while developing CourseBook. These challenges, while not necessarily problems that drastically changed our direction or progress on the application, required more in-depth critical analysis and planning to appropriately overcome.

The Parser

One of the principle challenges with the Parser was caused by the poorly formatted nature of the Registrar timetable, as well as the lack of coherence between the Registrar descriptions and timetable. As a simple example in extracting department codes from the main timetable page, the majority of codes can be identified in a line of HTML which looks similar to the following:

```
<a href="latn.html">Latin</a>
```

which a simple regular expression can identify, yet on occasion the line will exist in the format:

```
<a href="latn.html">Latin</a>
```

where on closer examination there actually exist a ‘/r’ and three ‘/t’ characters between the acronym and the .html. Now, not only does the parser have to take into account the new formatting, but it also has to strip away the ‘/t’s and ‘/r, otherwise the acronym itself will contain those characters. Such formatting issues with the HTML were fairly commonplace, and simple matching of the character string “<a href="[^.]*.” (treating . as a literal in this case) will generally extract the desired text.

More difficult was the task of assigning lines of text surrounding section information to the section to which it belongs. For example, the following is a line-numbered block of HTML extracted from the timetable for MATH 103:
Note, to begin, that it is a considerable challenge for a human being to understand what is expressed in this information. Does line 3 apply to the recitation on line 1, or does it apply to the lecture on line 8? What about line 10, then? As the parser examines the HTML, it must make decisions about whether information it is running across applies to section information it has already seen, a section yet to come, or to the entire course. The parser must determine which sections of MATH 103 should be assigned to GROUP 3, and must understand that the registration requirement applies to all below sections. The problem is that the criteria and caveats for every single possible course are so vast and varied that the parsing system quickly became incredibly complex. Up until the very end of development, quirks, unexpected formatting and text, and nonsensical registration HTML would occasionally surface as the wrong information being parsed for one or two sections in the entire registrar. It is very likely that future timetable releases may contain unexpected formatting changes, so while the parser is designed to be as robust as possible, it was also designed to be easy to alter.

Coherence problems, as mentioned previously, also had to be dealt with, like where courses exist in the timetable, yet are missing from the registrar itself. For instance, the recent change in the Computer Science department from CSE to CIS has left a gap in the registrar where no courses are listed as CSE on the time table, and the new CIS courses are not reflected in the course descriptions. Coherence problems such as this required manually identifying these discrepancies and allowing the parser to either ignore missing data, or patching the unexpected gap. Fortunately, not many of these problems existed.

The Database

The largest challenge in working with the database was the design of the schema for dealing with a MySQL database rated to handle relatively small amounts of data. In the design process, our challenge was to provide a structure that could be queried with as few joins as possible, but still provide storage for the data structures required in processing course information.

RPCs and Server-Side Development

When dealing with data passing between the client application and the server side, the largest problem was in the limitations of serializeability and parameterization. GWT mandates, as described
previously, that all information passing between the two parts be either a primitive data type (as defined by Java) or implement `Serializable`. Lists, which are considered `Serializable` by default, and higher level data structures, were also limited by an inability to utilize generics for type-safe interaction with the objects. To get around these restrictions, careful planning and wrapping had to be designed for passing course information returned by the server, is it could be easily unwrapped and understood by the client. Unfortunately, much of this interaction was very tedious, and often times could be rather challenging to implement efficiently.

**GWT and Client Side Development**

GWT was a double-edged sword in terms of what it has done for CourseBook. Many of the positive contributions of GWT to our project have already been discussed, but GWT itself provided us with some unique (and often frustrating) challenges. Neither of us was familiar with GWT from the start, and the documentation available for GWT, while fairly extensive in most parts, was sparse and unclear in others. Added to this was our decision to use the 3rd party GWT-EXT package of Widgets (www.gwt-ext.com), which provided even less documentation and troubleshooting references for us. To say that GWT has a steep learning curve would frankly be an understatement.

In addition to having to learn GWT were the frustrations of learning how GWT interacts with the three main web browsers when the code is compiled. GWT compiles down into HTML and JavaScript with CSS formatting applied to the page. Unfortunately (and notoriously), Firefox, Safari, and Internet Explorer do not always agree on exactly how to apply the same CSS to the same HTML. Something that displays perfectly centered and wonderfully in Firefox might not even initially be visible in IE, simply because of their different interpretations of relative positioning. These discrepancies made working with GWT for formatting an often painful and tedious process.

Besides learning to interact with GWT, one of the first technical challenges we ran into was efficiently retrieving and displaying huge amounts of data when users made expansive queries to the Registrar and Penn Course Review Search. For the Registrar search, we had originally planned to retrieve information on an ‘as needed’ basis, only querying the database for detailed information when prompted. However, we found that retrieving all course and section information at the time of the search was actually more efficient. While it makes the search take minimally longer, it ensures the application is quick and responsive when asked for detailed information.

When no (or minimal) search criteria was specified, a search could hypothetically take quite a while to return all relevant results. Instead of returning the thousands of sections that match this ‘criteria’, we decided to limit the results to the first 100 in such instances, and display a message encouraging the user to refine their search. While it is possible that someone would want to ‘browse’ the entire registrar from our site, we decided it was better to limit the results than to have the system hang while it searched and gathered data for excessive amounts of time.

Similarly, the Penn Course Review search returns an enormous amount of information, even if the user has specified some criteria to filter by. Once the information is returned, it needs to be processed to calculate averages for each of the categories, and load all of the information into the
required tables and forms, which are initially displayed as collapsed boxes with the course name and number as the title (unfortunately, the widgets we display these results in are not has efficient as the Grid widget, but the Grid was just not suited to our purposes). Therefore, while we still pull all of the information for the search back to the client side at one time, we wait until the user has requested to open the collapsed panel and see the detailed information for the course before doing all of the processing and layout. While this does increase the amount of time it takes open the panel, it dramatically improves the search time.

Displaying the Registrar results in the Grid widget was also a technically challenging task. The grid is designed to only accept text strings as data. While this is fine for most of our information, the ‘Add’ button needs to be an image, and the section number needs to be a hyperlink that will trigger the opening of the detailed information window. In order to accomplish adding the image and the hyperlink, we had to manually override the Grid renderer, and insert HTML into that field. While in the end the solution was fairly simple and elegant, it took an extremely long time to get it to render and display properly (GWT is not designed to let you edit the HTML directly, and often does unexpected things when you do). To be able to listen for a mouse click on either the button or the section number, we also had to create special click handlers that could tell which column the mouse clicked, and fire the appropriate response.

The original ‘detailed information’ view for each of the course, which has now been replaced with the popup dialog, was also one of the most technically challenging components. Originally an extra row was added after the final section for each course, and a ‘Show Details” button was rendered much in the same way before. However, this time we had to render the entire row to span all of the columns, a task that was not well documented anywhere. In addition, the custom click handler that was built had to handle collapsing and expanding the row to show or hide the details as necessary, and to dynamically insert and remove HTML (which turned out to be quite different from statically setting it as before) from this row. Although in the end the feature worked as planned, unfortunately user tests showed that people could not find the button, or completely ignored it, even if prompted to find detailed information for the course. Multiple users trying to click on the section number to retrieve this information (which at the time did nothing) prompted us to scratch our original design in favor of the overlay popup dialog.

For the schedule building utility, the largest challenges were in developing the multitude of user-facing features. Our project was less concerned with building automated and highly technical scheduling tools, and more with providing an easy framework for our users (the students) to interact with their course selections, build them into schedules, and deal with conflicts and missing requirements. Each of these features, no matter how small or frequently used, often required a considerable amount of supporting code. Tasks such as examining the schedule for missing recitations or lectures based on a course’s requirements and then displaying a warning are surprisingly complicated and intensive. The challenge of designing a Schedule object to make as many of these small tasks as easy as possible was considerable, and very important for easing development.
3.6 Block Diagrams

LEVEL 0 – ABSTRACT

LEVEL 1 - DETAILED
4. Non-Technical Approach

4.1 Non-Technical Challenges

While CourseBook was a technically demanding endeavor, many of the problems we faced involved the real world implications and considerations involved in developing a large application. One of the biggest challenges was in convincing the University to give us access to the course databases to save us the task of parsing the registrar to extract our data. This was actually such a challenge that we failed at it quite stupendously. Initial talks with University officials were met with cold indifference and an unwillingness to engage in conversation, and when pressed, led almost to open hostility towards the subject. Even Penn Course Review, which is run by the Undergraduate Assembly and in which we have a personal contact, was unable to offer us any assistance in the matter. Needless to say, and likely due to their database schema being privately designed (and incredibly old), the University was unwilling to give out access to even a copy of the database for reading purposes. Given past attempts by other groups, this was not wholly unexpected or unprepared for, as we were already well into development of our parser by the time all possible channels were explored and denied.

Another bump in the road involved the University’s WEBSEC application for PennKey authentication of online applications. WEBSEC is a University developed framework for requiring all users of a WEBSEC enabled application to authenticate themselves with their PennKey before using the site. However, our initial request to obtain authorization to develop with WEBSEC was denied by ISC (for reasons unknown), and so development proceeded under the assumption that there would be no PennKey authorization, and we would devise our own system of registration. However, late this semester when we attempted to acquire a notice of verification of denial to be able to present it as the reason for developing the application in the manner we did, ISC surprisingly decided to approve our application for WEBSEC. Unfortunately, this occurred so late in the development process that incorporation of WEBSEC simply was no longer possible given the time constraints. Still, this added the extra challenge of devising our own system of user registration, something we had not planned on having to do at the beginning.

A nontechnical challenge of the project, which also manifested itself technically, was in having to deal with the extremely large amount of information available to us. The information retrieved from registration involved over 20 different fields, and limited screen real estate forced us to be selective in what we could and could not display. Decisions had to be made regarding the relative importance of information, such as the maximum enrollment of the section versus difficulty ratings. On the My CourseBook page, only so much space existed in the CBBox objects for us to summarize a section, and we had to determine which fields were most important in this context. Even in the Registrar search form
we were forced to prioritize for screen real estate by eliminating the less relevant search criteria, and carefully organizing the fields to avoid clutter.

It should be noted here that the visual design itself, as described above, was a challenge to understand and get correct. Neither of us have any formal training as designers or artists. We were trained as engineers to provide cold, hard functionality with little, if any, consideration for usability and aesthetics. Forcing ourselves to rethink our methods and produce an application, which was more than purely functional, was incredibly challenging and often took far longer than we expected it to. We had to learn everything from how to choose appropriate color schemes to how to intuitively layout buttons and search fields on a form. As we learned, we went from coding first and deciding on layout later, to forming rudimentary mocks in MSPAINT to guide us, to eventually creating full scale, production quality mocks in Adobe Fireworks. Many of these tools were foreign to us, but with perseverance, and a bit of trial and error, we were able use the effectively in creating a fluid and user-focused design for our site. While we cannot claim to be anything more than mediocre at this point in our design skill, we now know immeasurably more about the subject that we ever expected to have to learn at the outset. Training ourselves to examine both the technical and visual problems presented in developing new features was probably one of the most ‘real-world’ relevant lessons we could have hoped to learn from this project.

The final non-technical challenge, and possibly the most important was that the University decided to release a new Course Registration and Schedule Planning utility in early March, much to our (and everyone else’s) great surprise. Initial talks with the University had led us to believe that while they were (and had been for a few years) thinking about updating course registration, they gave no indication which lead us to believe they were actually going to do it this year, and ignored our repeated attempts to show them the potential value of the system we were planning. When the University then released a system which was incredibly similar to our concept of course registration (a coincidence? You decide) we decided to rethink our approach. While we could still provide a system with more and better functionality than their new release, we decided to put a strong focus on building a system that more accurately reflected what the students wanted in course registration. To accomplish this, we were presented with an additional challenge: collecting feedback from students all over the University about our project and centering further development on their preferences.

4.2 Collecting and Analyzing Feedback

By the time the University released their new registration system, CourseBook’s fundamental functionality was largely completed. Searching and schedule building existed in rudimentary forms, but much of the layout and design was still lacking. Since, as is the case with many large-scale applications, the original feature set far exceeds what (for just the two of us) we would feasibly be capable of implementing in a few months, we needed some way to prioritize the features to implement. We sent out a survey, through various listserves we belong to, containing a few simple questions to gauge what the general student body required of registration (for graphical survey analysis, see Appendix B).

One of the first questions in the survey addressed our concern in determining the most relevant search criteria for the Registrar search. By providing an entire host of options for search, based on the
University’s search criteria and our own data fields, and asking students to rank how important each was, we were able to determine which search criteria students considered the most, and least, vital. Results indicated that students most strongly desired searching by Instructor, the time of day and days of the week the course met, the requirements fulfilled by the course, and Penn Course Review information. Criteria that were surprisingly less important were the course program (CGS, Ben Franklin Scholar, etc) and first day on which the course meets. We used this feedback to finalize the search fields that appeared on the Registrar search form.

To help determine the best formatting for our search results, two different versions of Registrar results were presented. One was a screenshot of the University tool’s search results (which we used as a baseline to determine how ‘good’ our feedback was), and the other was a modification on our current system, incorporating various ideas we wanted to gather feedback on. Users gave their opinions of each design, by rating them on five visual criteria. It should be noted that we did not show the images to the users at one time, but instead presented them with one, followed by the other on the next page. This was to hopefully prevent direct comparison, and instead judge the individual merits of the two designs.

The final design for the Registrar search results in CourseBook was based largely on the positive feedback the mockup received. Users responded that it was easy to understand, visually pleasing, relatively uncluttered, contained the essential information they were looking for, and was organized appropriately. They indicated that the University’s system, while almost equally functional and logical, was less visually attractive and far more cluttered. It seemed that students preferred the grouped-by-course style of CourseBook, and our decision to display less information in the results, instead moving the ‘extra’ info to the detailed course window.

One question that produced some surprising results was a ‘general preferences’ question that attempted to gauge interest in different potential features. Many of these features were high on our priority list, as we expected them to be highly demanded by students. However, our survey showed that some of the features we listed were relatively unimportant to students. One particular example was in sharing schedules, a feature that, at the inception of CourseBook, we felt would be a driving motivator to use the site. However, 84% of those who responded indicated that sharing a schedule or looking at a friend’s schedule was ‘not important’ to ‘somewhat important’ to them in a scheduling utility. It was largely based on this feedback that we lowered this feature’s priority, and eventually ended up making the decision to not incorporate schedule sharing into this version of CourseBook.

To help refine the look and feel of CourseBook, usability tests were conducted with a considerable number of friends and acquaintances throughout the development process. Students were presented with a page of the website, and asked to try to accomplish a specific task, such as searching for classes, adding classes to their CourseBook, finding out the detailed information for a class, and building a mock schedule. During the process, we encouraged students to think out-loud, and give reasoning for why they chose to click the buttons they did, or to explain elements that they found confusing. For each test subject, we noted both what he or she found easy and difficult about particular features, and attempted to modify our design to incorporate this feedback. For instance, on the Penn Course Review search, one of our subjects failed to realize that the drop-down menu controlled which
semester of information was being viewed at the time. When asked why this was, she replied with “It’s hard to distinguish it from the background and sides here. Can you make it look more like it’s attached to the box?” This type of feedback guided our modifications to the layout and design for many aspects of CourseBook.

One of the last major feedback related changes had to do with our planned automatic schedule builder. During the user tests described above, almost every single one of our test subjects ignored the option to have the system automatically build a schedule for them. When asked if this was because they did not understand what the option did, they generally replied in the negative, stating that they preferred to manually control what was going on their schedule. Two of the subjects called the feature “unnecessary” and that at most all they wanted was an option to add all courses at once so they could remove extras or conflicts on their own. This general indifference towards an automated schedule building tool caused us to lower the priority of fully developing the algorithm, and eventually the feature was discarded.

4.3 The Future of CourseBook

Continued Development

Although we are incredibly pleased with the final outcome of our project, we are well aware that there is room for improvements to both the interface and the functionality of this system. For this reason, we have decided to continue working on additions to CourseBook as our final project for CIS 399 (Human Computer Interaction with Jeffery Nimeroff). The final project for that class is supposed to be a web application with a focus on usability, design, and a social networking component. We feel that these categories are exactly in line with the areas of CourseBook we would like to fine-tune the most.

Our first goal is to continue to work on the user-interface design. We hope to overhaul the color-scheme and change it from its current corporate style, to a more fun and ‘student friendly’ design. We also hope to continue to gather feedback from fellow students on ways in which our application could be made more user-friendly. While we have worked exceptionally hard on both of these components all year, we hope to be able to apply the knowledge and techniques we have recently acquired in HCI to our application. Since the application is far more complex than anything we could have created as a final project for CIS 399, we hope it will be a greater exercise of the skills we have learned.

In addition to the enhancements to the user-interface, we would also like to add a ‘social’ component to CourseBook. Although we would have loved to include these features in our Senior Design project, they are out of the scope of our current feature set. First, we would like to add the ability for students to add comments about classes, and display these alongside the Penn Course Review data. This will be a challenge because we will need to rethink the way we display Penn Course Review information to include the comments, develop an easy and intuitive way for students to leave feedback, and determine ways that users can search for this new information. We would also like to add functionality to export a schedule. Right now, we are looking to allow users to ‘link’ to a schedule which will be a static HTML page separate from CourseBook’s main application. This will allow users to share
their schedule with others without the recipients needing to log into CourseBook. This was originally to be a feature of CourseBook, but it was unfortunately too low on our priority list to make it into the final product.

**A Renewed University Interest**

In the beginning of the year, we continually pitched our ideas to the University, nearly begging them to let us develop a tool to solve their course selection problems. However, we were told that they were not interested in student projects, and could not help us in any way.

However, in the last few weeks, there has been a renewed interest from the Student Committee on Undergraduate Education (SCUE) and the Undergraduate Assembly (UA). We have been contacted by the Chair of SCUE, who is very interested in learning more about our findings on the preferences of students. While we have been told that the University is still unwilling to adopt our system, especially since they have now invested in a system of their own, they are eager to look to our project for ways in which they can improve the Penn-In-Touch application. Although this is slightly less gratifying than them actually using our system, it is satisfying to know that our tool will have a positive and lasting impact on almost all of the students at the University.

### 5. Conclusion

In two semesters, restricted from access to University resources and data, CourseBook has managed to develop from inception to an arguably better product than the University and a team of professional programmers took more than ten years to devise and implement. We have developed a database of course information almost assuredly better constructed than the 1970s ‘ETABASE’ data warehouse which currently houses registration information. We successfully managed to develop a robust and relatively quick parser to scrape awfully formatted HTML for insertion into our database. Most importantly, we have developed a visually pleasing and easy to use application, that is in line with the current ‘Web 2.0’ design trends. It is an application that is not only feature rich, but designed for the students, by the students.

Equally noteworthy, CourseBook has already provided, and will continue to provide feedback to the University about student preferences and registration habits. The student bodies which have taken notice of and interest in CourseBook, desire not only our survey results for forming their proposals to the University as they continue to develop new and exciting ideas for the future of Penn-In-Touch, but to use the application itself as an example of what course registration should aim to be. CourseBook, while no longer breaking new ground, has still paved the way and provided a lasting example to the University.
The developmental road to CourseBook was not always as easy as expected. Many of the aspects of web design, interface layout, and user interaction were far more difficult to understand, even at a fundamental level, than we could have anticipated. In the same vein as design considerations, we could not have anticipated the level of technical difficulty in formatting and adjusting layouts to render the site pleasantly across the different browsers. Each browser possessed unique and sometimes befuddling quirks, many of which took hours to find, let alone fix, the source of the rendering problem. Finally, a project like CourseBook is never complete. There is always one more feature, one last little thing which we could have added, but we had to make the cut somewhere. Deciding what we simply either did not have the space for, or did not have the time for was incredibly difficult in itself, but failing to make those decisions would have resulted in a far more mediocre project. We decided to focus on making the core set of features as functional and user-friendly as possible, as opposed to adding tons of ‘not quite ready,’ low-priority features.

On a more positive note, we found it was easier than we had expected to acquire feedback from the student body. Never had we expected that the students would be so responsive to our questioning and surveys. It surprised us time and time again when a student was genuinely interested in our project and wanted to help us make it better by providing incredibly constructive and detailed feedback of their interactions and thoughts in design. It was encouraging to know that the student body seriously cared about course registration; either that, or they just really thought we were swell people.

Given the successful outcome of the project, in retrospect there does not appear to be that much we would have done differently, apart from one major thing: Armed with the knowledge that the University was going to unexpectedly release an update of course search and registration from the outset, we would have begun the user-centric phase of CourseBook’s development much sooner in the process. This would have not only given us much more time to learn the difficult intricacies of design and acquire more outside help and resources for that component, but would have frankly prepared us for the gut-wrenching blow of discovering the update for the first time. We can assure you, it was tragic. Overall, however, CourseBook has been an incredible learning experience, and we would have preferred no other project in its stead.
6. References


In this March 2008 Daily Pennsylvanian article, Bell describes the current state of the University registrar and details the functionality of the University's proposed system that would launch in the middle of March. The article states that Anthony Maggio, the Junior Class president, and UA members Michelle Tandler, Alex Flamm, both Undergraduate Assembly members, worked directly with Student Registration and Financial Services and the Information Services and Computing office on the project, which had been in development for over a year. While the article states that the University conducted focus groups with students, some of the feedback we received is in direct conflict with many of the design choices they made. Overall, the article is objectively written, though most of the quotes are by those in charge of overseeing the development of the system and are therefore very positive towards the system. This article was also the first time we heard that the University was releasing a tool of their own, even though we had been in direct contact with two of the organizers.


In this follow-up piece to "For Penn In Touch, a refresh is on the way," Bell takes a look at the initial reaction to the University's course searching and schedule planning tool. Many students liked the ability to search by requirements, and to visually see conflicts in classes, which reinforced our decision to include these features (though they were high on our priority list and would have been included regardless). Although the responses are mostly positive saying that the tool makes course planning more "convenient" and "less time consuming," students expressed disappointment with the inability to see Penn Course Review information on the site, and having to still take the extra step to manually register classes on Penn In Touch. The article states that since the launch (approx. 7 days prior), almost 4000 students logged onto the system, and about 2000 created mock schedules. This represents a significant portion of the student body (especially if you do not include seniors class who did not need to register for classes this semester), and this shows the incredible demand for a course search tool.


PickaProf.com was originally developed by Chris Chilek and John Cunningham as a tool for students to rate professors, much like Penn Course Review. PickAProf.com contracts with
universities to provide enhanced functionality to course planning and registration. It began as a tool for students to rate professors, much like Penn Course Review, but has expanded to include schedule and degree planning. Users can browse courses, see professor ratings and grade histories (i.e. how many A’s or F’s a professor has given in past years), and create a timetable with their requested classes. Additionally, users can search for textbooks for their classes. Adopting such a system would have been a huge step in the right direction for Penn, especially considering the rich feature set. PickAProf.com claims to have over 180 Universities signed up, showing that there is a strong demand for these types of systems. However, since this is the company’s website, the information is obviously going to be positively biased. Regardless, the feature set is impressive, and gave us a long list of ideas that we could use as inspiration for our product.


Registroogle was a senior design project by Andre De Clercq and Shreyans Bhansali in Spring 2004. They set out to build a system where you could search for classes, plan your schedule, and ‘one-click’ register, instead of going through the lengthy registration process on Penn-In-Touch. However, the University denied them access to both the course database and the system which registers students for classes. In the end, their project was a web based application that allowed users to “search through all the courses offered at Penn using a combination of over 10 different criteria, read details about the course, and build a schedule that tracks conflicts and statistics for the proposed schedule” (De Clercq and Bhansali, 2004).

This project excellently demonstrated the need for a better course planning tool at the University. Unfortunately, it was not implemented in a sustainable way, and disappeared when the creators graduated. It also lacked an aesthetically pleasing design, as it was far more focused on pure functionality than usability. The greatest lesson learned from this paper was that the University was unlikely to grant us access to the course registrar database, and therefore we decided to start building a parser as soon as possible. It was also interesting to see the criteria that they allowed users to search by, and ultimately ‘set the bar’ for what would like to achieve with CourseBook.


In this Daily Pennsylvanian article from fall semester 2007, Kaplan describes the ever growing need for an update to the aging Penn In Touch system. She talks with Alex Flamm, who states that they are working with the ISC, who have been "receptive" to the idea, and the Registrar, who have been far more reluctant to updating the system. Flamm is quoted as saying, "Their basic opinion was that they'll replace it, but not for another four years or so." This article was published days before the original draft of our paper was due, and it further affirmed our
ideas that CourseBook would be a welcomed relief from the tedium of course planning. It also increased our confidence that the University might be willing to work with us and support our project, as they already seemed to be looking for a solution. This article also made the introduction of the University's new tool in March 2008 even more shocking, as we had thought nothing would be done for ‘four years or so’.


Developed by Steve Kirkham at the University of Alberta, BearScat makes it possible for users to search for classes based on department, course number and subject, and plan a schedule to fit those classes. It allows students to directly register for these classes, receive notifications when a full class has a space open, and add/drop/swap classes. It also provides a listing of required course texts, and access to professor ratings. Although this application was developed independently by a student, the University has provided access to its databases and has even funded BearScat in recent years. According to the BearScat website, last year the application “served 10.6 million hits, 1.2 million transactions to [the University of Alberta’s registration system], 150,000 notifications, and 135,000 designed schedules” (Kirkham, 2006). This project is a strong example of the increased value potential for a system that is University supported. In speaking with Kirkham, I found that while the University was reluctant to work with him at first. Once he had a functioning website however, they were much more willing to support it. This is similar to the situation we are in now, as the University has begun to show interest in our application, though it is unlikely that they will adopt it as their main tool.


Registrar Search developed by Gayle Laakman in 2005 (Laakman, Registrar Search, 2005). Her tool scraped all of the course information from the Penn Registrar and allowed users to search for courses based on time, days, subject or professor. Although her application lacked rich functionality, it was another demonstration of the desperation of students for a course searching utility. It was a purely functional tool, with just a form and a listing of search results, but it was still infinitely better than the University's non-existent system. Unfortunately, Gayle has since graduated, and has moved her site from the www.glaak.com domain name, rendering the site unavailable. However, we were fortunate enough to get in contact with Gayle and she graciously showed us her (now fairly outdated) parsing utility, which we referenced as a starting point for the core of our parser. Gayle's site also provided a basic set of search criteria that we could base our original search form off of.


The Penn Course Review was a senior design project by Howie Vegter and Steve MacCrory in 2002 (MacCrory & Vegter, Penn Course Review: Homepage, 2002). Vegter and MacCrory
managed to form a partnership with the Student Government and Undergraduate Assembly who agreed to support their project. The project was initially deemed a huge success, as it provided a much needed upgrade to the nearly-unusable solution the University had originally created. However, over time it has become a burden as the University continuously tries to find people to maintain and support the site. Vegter and MacCrory’s project gave us some hope for obtaining University approval, but in the end, its non-sustainability may have actually hurt us as the University seems unwilling to support future senior projects.


ClassBuster was a tool developed by University of Pennsylvanian student Danny Panzer that allowed users to repeatedly check for openings in full classes and send an email or text message when a class became available. However, the University explicitly warned students against the use of this program, as the application was not authorized to access Penn-In-Touch, and it was eventually taken offline. Although our system will not be able to provide this functionality for the same University mandated reasons, it provided us with some important lessons about dealing with University policy.


In late March 2008, the University unexpectedly released a Course Search and Planning tool. This tool allows students to search the Registrar Timetables, as well as view detailed information about classes (description, requirements fulfilled, prerequisites, etc). The tool also allows students to create mock schedules, but registration must still be manually completed through Penn-In-Touch. The application does not provide access to any data from Penn Course Review. While the tool does include many of the same features we proposed for CourseBook, the site is both aesthetically lacking, and can be confusing to use. The site seems to have been designed with minimal student input, as much of the search criteria and information displayed is not relevant for most students planning their schedule. With such vast resources, it is disappointing that this is the best solution the University could come up with to the glaring problems of course selection. The release of this tool however, did serve to partially change the focus of our project from being a purely feature driven website, to a site focused on being more relevant to students and usable.


The Critical Writing department provides a tool to search writing courses by term, program, department, instructor, meeting day, meeting time, and keyword. This is an excellent tool for finding writing courses, and again demonstrates the need for easily searchable course information. Unfortunately, since it makes only writing courses available for browsing, it is of little use to the overall student population. We originally considered adding this tool to our project, but we felt that in the end it would be a redundant feature and since most people only
search for a writing class one semester, that the Critical Writing Department's tool was more than adequate for this purpose.


Prior to March 2008, the University Registrar was the only University supported source of course timetable information. One page provides course descriptions and prerequisites, and another provides course timetables for the semester. There is a link to the Penn Course Review, a site where professor and class evaluations are stored. Registering for classes requires logging into Penn-In-Touch, and manually entering the desired classes (they must be written down or looked up beyond the registration utility). During advanced registration it fails to inform of any conflicting selections, nor provide any visual representation of a requested schedule, apart from a list. It has long been recognized that the Registrar is a poorly designed maze of links and pages that makes course planning a nightmare. This was the driving force behind our project, as above all, we wanted to make course registration an easier and more efficient process for everyone at the University.


In this slightly outdated article, Yahalom explores some of the third party alternatives to Penn Course Review that exist at other schools, specifically PickAProf.com, which had recently been adopted by Brown. Once again, it is stated that the ISC is looking to launch a revamped site, but no specific information is given. Gabe Kopin, the chairman of the Student Committee on Undergraduate Education does say that the University has no plans to outsource the task however. He states that PCR is unique in that it has a "more thorough and diverse representation from the student body that its competitors do not." Although this article did not have much of a direct impact on our project, it was an interesting insight into why the University has resisted adopting a 3rd party tool, and insists on building its own system.


In this article, Yahalom talks about the student created program ClassBuster, and the University’s response. ClassBuster was designed to alert students when a class they wanted had a spot open. However, ClassBuster did not have authorization to use PennKeys, and therefore the University warned against it, stating that the security of your account. This article served as a warning to us that we should not require PennKey login, but instead just use an email address and a completely separate password. While it would have been nice to authenticate with a PennKey, we feel that a upenn.edu email address is enough validation for our purposes.
The above graph is extrapolated from the parts of the survey results dealing with user preferences on search fields. It has been ranked in order of what students considered least important to most important. This analysis was valuable in helping us to determine which search fields would be included in CourseBook’s registrar search.
The above graph came from analysis of user feedback regarding their opinions on different design criteria applied to CourseBook and the University’s search results. This feedback helped to confirm our design decisions as subjectively superior and further guide development.
The above graph, similar to the Analysis of Search Criteria graph, attempts to rank the students’ preferences in general functionality of a course registration system. Again, this analysis was highly important in deciding which criteria eventually would be included in CourseBook.
Appendix B  The Database Schema and Column Descriptions

CREATE TABLE Courses (  
    CourseID varchar(15) NOT NULL,  
    Department varchar(6) NOT NULL,  
    CourseNumber int(5) NOT NULL,  
    CourseTitle varchar(50),  
    Description varchar(500),  
    Prerequisites varchar(100),  
    PRIMARY KEY (CourseID),  
    FOREIGN KEY (Department) REFERENCES Departments(Code)  
)

CourseID: Randomly assigned numeric string, unique for each course.  
Department: Acronym for the course’s department (e.g. ‘cis’)  
CourseNumber: The course number (e.g. 120)  
CourseTitle: Course title given to the course in the timetable (e.g. Senior Project)  
Description: Long, textual description given to the course in the registrar.  
Prerequisites: Any text found in the description which comes after the word Prerequisite(s) (e.g. “CSE 320 or equivalent)

CREATE TABLE Sections (  
    SectionID int(5) NOT NULL,  
    CourseID varchar(15) NOT NULL,  
    SectionTitle varchar(50),  
    Professor varchar(25),  
    MaxEnroll int(5),  
    Dates varchar(25),  
    CourseType varchar(10),  
    Credits varchar(5),  
    Grp varchar(5),  
    ExtraInfo varchar(200),  
    PRIMARY KEY (SectionID, CourseID),  
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)  
)

SectionID: Section number for the course section, e.g. 401  
CourseID: ID of the course this section belongs to  
SectionTitle: Name of the specific section, applicable when different sections have different topics.  
Professor: Name of the instructor for the section.  
MaxEnroll: The maximum enrolled number of students in the section.  
Dates: Applicable for graduate level courses which may not meet all semester.  
CourseType: Section classification, e.g. LEC or LAB
Credits: CU awarded for the section.
Grp: Applicable for sections where registration is required for a LEC and REC in the same group.
ExtraInfo: All extraneous information on the timetable unable to be grouped into a category in the database.

CREATE TABLE Reviews (  
Department varchar(6) NOT NULL,  
CourseNumber varchar(5) NOT NULL,  
SectionID int(5) NOT NULL,  
Semester varchar(20),  
CourseTitle varchar(50),  
Lecturer varchar(50),  
Ret varchar(7),  
RateCourse varchar(5),  
RateInstr varchar(5),  
RateDiff varchar(5),  
FOREIGN KEY (Department) REFERENCES Courses(Department),  
FOREIGN KEY (CourseNumber) REFERENCES Courses(CourseNumber),  
FOREIGN KEY (SectionID) REFERENCES Sections(SectionID)  
)

Department: Acronym for the department (e.g. ‘cis’)  
CourseNumber: Number for the course (e.g. 120)  
SectionID: The sectionID of the course (e.g. 410)  
Semester: The semester from which these ratings came (e.g. Spring 2004)  
CourseTitle: The name of the course at the time.  
Lecturer: Who taught the course.  
Ret: The return on the course ratings from the students. The number returned out of the total possible.  
RateCourse: Average rating given to the course by the students.  
RateInstr: Average rating given to the instructor by the students.  
RateDiff: Average measure of difficulty of the course for the semester.

CREATE TABLE ReviewDetails (  
Department varchar(6) NOT NULL,  
CourseNumber int(5) NOT NULL,  
SectionID int(5) NOT NULL,  
Semester varchar(20) NOT NULL,  
CourseTitle varchar(50) NOT NULL,  
Professor varchar(50) NOT NULL,  
CQuality varchar(5),  
PQuality varchar(5),  
Difficulty varchar(5),  
Communicate varchar(5),
Accessible varchar(5),
Readings varchar(5),
Learned varchar(5),
AmtWork varchar(5),
RecommendMajor varchar(5),
RecommendNon varchar(5),
Interest varchar(5),
PRIMARY KEY (Department, CourseNumber, SectionID, Semester, Professor, CourseTitle)
)

Department: Acronym for the department (e.g. ‘cis’)
CourseNumber: Number for the course (e.g. 120)
SectionID: The sectionID of the course (e.g. 410)
Semester: The semester from which these ratings came (e.g. Spring 2004)
CourseTitle: The name of the course at the time.
Professor: Who taught the course.
CQuality: Average rating given to the course by the students.
PQuality: Average rating given to the instructor by the students.
Difficulty: Average measure of difficulty of the course for the semester.
Communicate: 0-4 Ranking of course criteria
Accessible: 0-4 Ranking of course criteria
Readings: 0-4 Ranking of course criteria
Learned: 0-4 Ranking of course criteria
AmtWork: 0-4 Ranking of course criteria
RecommendMajor: 0-4 Ranking of course criteria
RecommendNon: 0-4 Ranking of course criteria
Interest: 0-4 Ranking of course criteria

CREATE TABLE Departments (  
    Code varchar(6) NOT NULL,  
    Title varchar(50),  
    PRIMARY KEY (Code)  
)

Code: The course code acronym (e.g. ‘cis’)
Title: The proper name for that department (e.g. ‘Computer Science’)

CREATE TABLE Meets (  
    CourseID varchar(15) NOT NULL,  
    SectionID int(5) NOT NULL,  
    Weekday varchar(5) NOT NULL,  
    StartTime int(4) NOT NULL,  
    
)
Leicht, Moore, 39

```sql
CREATE TABLE Satisfies (  
   CourseID varchar(15) NOT NULL,  
   SectionID int(5) NOT NULL,  
   Requirement varchar(50),  
   PRIMARY KEY (CourseID, SectionID)  
)
```

CourseID: Reference to the course to which this entry refers.
SectionID: Reference to the section to which this entry refers.
Requirement: Requirement which this course satisfies e.g. ‘GEN REQ IV’

```sql
CREATE TABLE Users (  
   email varchar(50) NOT NULL,  
   password varchar(200) NOT NULL,  
   question varchar(200),  
   answer varchar(300),  
   PRIMARY KEY (email)  
)
```

e-mail: The user’s email address
password: An encrypted copy of the user’s password
question: A security question used to verify identity for password reset
answer: The encrypted answer to the security question

```sql
CREATE TABLE Schedules (  
   schedID int NOT NULL AUTO_INCREMENT,  
   email varchar(50) NOT NULL,  
   name varchar(50) NOT NULL,  
   PRIMARY KEY (schedID),  
   FOREIGN KEY (email) REFERENCES Users(email)  
)
```

Weekday: The day(s) of the week relevant to this entry e.g. ‘MWF’
StartTime: The start time (military time) of the course on the given day(s) e.g. 1030
EndTime: The end time (military time) of the course on the given day(s) e.g. 1200
schedID: A unique ID number for each schedule
email: The user to which this schedule belongs
name: The schedule name

CREATE TABLE CBookCourses (  
cBookID int NOT NULL AUTO_INCREMENT,  
email varchar(50) NOT NULL,  
Department varchar(6) NOT NULL,  
CourseNumber int(5) NOT NULL,  
SectionID int(5) NOT NULL,  
Color varchar(20),  
PRIMARY KEY (cBookID),  
FOREIGN KEY (email) REFERENCES Users(email)
)

cBookID: A unique ID number for each course entry
email: The user which added this entry
Department: The department of the course (e.g. ACCT)
CourseNumber: The course number of the course (e.g. 101)
SectionID: The section number of this section (e.g. 001)
Color: The color configured by the user for displaying this course on a schedule

CREATE TABLE ScheduledCourses (  
schedID int NOT NULL,  
cBookID int NOT NULL,  
PRIMARY KEY (schedID, cBookID),  
FOREIGN KEY (schedID) REFERENCES Schedules(schedID),  
FOREIGN KEY (cBookID) REFERENCES CBookCourses(cBookID)
)

schedID: The unique ID number for the referenced schedule
cBookID: The unique ID number for the referenced course