April 2007

Enhancing OfCourse

John W. Furman
Worcester Polytechnic Institute

ENHANCING OFCOURSE

A MAJOR QUALIFYING PROJECT REPORT
SUBMITTED TO THE FACULTY OF WPI
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE BY:

JOHN FURMAN

ADVISED BY:
PROFESSOR KAREN A. LEMONE

Abstract

This project made several key improvements to the distance education application OfCourse. Research was conducted into the field of distance learning and semantic searching. The software structure was reorganized from a monolithic entity to a plugin architecture. Several important security vulnerabilities were recognized and fixed. Many of the tools used in OfCourse were redesigned or replaced entirely. Finally, a discoverability search tool was added to provide a means for users to perform system-wide searches of the information contained within OfCourse.

Acknowledgements

I wish to express my utmost appreciation to the following individuals:

Professor Lemone for her advice, guidance, feedback and support throughout the course of this project and for loaning me her laptop to use during this project.

Seth Hunter for his participation in the testing phase of this project, as well as his help on various issues of the design and implementation of the same.

The Library Staff of Gordon Library for their assistance in locating the necessary research materials for this project.

The people at the writing center of WPI for their help in proofreading this report.

Heidi Rose, Manny Riley and Patrick Swanson for their help in testing this project.

The developers of Promia, an open source web based file sharing program, which was used as one of OfCourse's plugins.

The developers of PAM, an open source web based quiz system, which was used as one of OfCourse's plugins.

Matthew Smith, a friend and colleague who will always be fondly remembered.

Table Of Contents

Table Of Figures

Table Of Tables

Executive Summary

This project enhanced a web-based distance education application called OfCourse. OfCourse was a fully functional software package from the beginning; However it needed some critical and major improvements to the functionality from the points of view of the students, instructors, and developers who intended to interact with the application.

Before attempting to do this, I discovered many techniques on how to maximize learning by students and the teaching efficiency of instructors through the careful and appropriate craftsmanship of distance education software. For example, tools such as a discussion boards should be designed to be structurally and functionally similar to web based applications that users are already familiar with. Therefore, I looked to several well-known web-based applications when redesigning the user interfaces of OfCourse's tools.

However, the improvements that OfCourse needed went far deeper than the user interface alone. OfCourse was constructed over several different MQPs and IQPs by many different teams. As such, the application structure was a loose knit patchwork of several different pieces of relatively monolithic code. There were essentially two choices at this point: Add my own improvements and changes to the monolith, or rearrange the application into a more organized software structure. I chose the second route. Because OfCourse was by and large a collection of relatively distinct sub-applications, such as chat rooms, discussion boards, file exchanges and the like, I decided the best approach was to reorganize the software into a plugin style architecture, consisting of plugins and APIs. This change vastly simplified the improvement of existing components and the addition of new ones, while breaking tight software cohesion and improving efficiency and maintainability. Also, this fundamental change allowed a single install of OfCourse to handle a theoretically infinite number of simultaneous courses, as opposed to the original application that could only handle one course per install.

While doing this restructuring, several major security holes were discovered in the system. These holes would allow a cracker to potentially invade the system and alter and/or destroy data. Most of these vulnerabilities centered around the login page and the

cookies that were sent out to the users browser by the web server. I used several web-based application security techniques to design and implement a rolling-code based cookie generation algorithm for secure cookies. I also modified the login page to run over Secure Socket Layer (SSL), to prevent user's accounts from being stolen. This changes vastly improved the security of OfCourse, making a malicious compromise of the application a far less than trivial task.

With a well-structured and secure code base now firmly in hand, the project shifted gears to specific improvements at the plugin level. The old file exchange, which was extremely cumbersome to use, was replaced with a very intuitive, user-friendly third party tool taken from the open source community. Integrating this outside piece of code was surprisingly easy, since OfCourse now supported a plugin style architecture. Another tool that was in need of attention was the quiz system, because it was badly broken to the point of being non-functional. Once again, I found a suitable quiz system from the open source community that was subsequently integrated into OfCourse as a plugin.

While some of the existing plugins were badly in need of improvement, perhaps the biggest discrepancy that separated OfCourse from being as powerful as it could be was the absence of any electronic search facility. Therefore, designing and implementing an adequate system-wide search tool was a major focus of this project. Since the application contained several plugins, each with its own type and method of storing and processing data, finding a method to seamlessly search them all accurately and efficiently was anything but trivial. To find a solution, I looked to the work of Watson and Wiley, two pioneers in a technique called Discoverability Search. Discoverability Search is specifically suited toward searching over many heterogeneous data sources (such as corporate data warehouses and the world wide web). As such, it was a natural choice for the technique to use for the design of OfCourse's search system. To accomplish this, I implemented a semantic structure, called a "metadata space" in the words of Watson and Wiley, to structure and organize the myriad heterogeneous data contained within OfCourse. I then selected, implemented, and deployed the appropriate algorithms to compare search strings and search criteria to the metadata spaces and produce search results ranked in order of their discoverability (relevance). This form of search was implemented over the discussion board, file exchange, chat room, course calendar, user

directory and static web-based course content. This provided an easy and intuitive means for a student or teacher to quickly and effectively find information of interest within OfCourse, through the use of a familiar user interface in the form of a search box (eg: Google). I also provided advanced search features and controls for users to further refine the selectivity of their search as necessary.

Software testing was indeed mandatory with all of the changes made to OfCourse throughout this project. Therefore, I designed three specific regression and integration tests according to the IEEE 824-1997 standard. These tests were then deployed to several volunteer human testers. I used their feedback to locate and fix several bugs as well as improve the aesthetics and ease of use of the user interface. As a result, future users of OfCourse can be confident that the system has been thoroughly tested and that a high level of quality and reliability has been achieved.

The efforts of this project vastly improved the functionality, usability, security and the information discoverability of the distance education tool OfCourse. The result is a very intuitive web based teaching tool that can be easily deployed on a wide variety of platforms, such as web hosts and web servers, and can be seamlessly integrated into any existing online curricula the instructor might have.

1. Introduction

Distance education is the teaching of and learning from course material in which no physical contact between the teacher and student occurs. As a result, many methods of long distance communication have been used to connect students and teachers together to form a positive and productive learning environment, despite their physical separation. Before the mainstreaming of the Internet, distance educators and students rely on such technologies as snail mail, fax machines, and telephone calls to bridge the gap (Pallof & Pratt, 1999). However, with a personal computer in almost every home that usually has some kind of Internet access, the potential for major new advances and ideas in distance learning became possible. These new technological powers also brought new technological struggles along with them. Among these challenges were the questions of which medium would be most appropriate for maximizing educational value, how to best transform any existing course material into electronic format, and how to design a computer system that is both a powerful manager and disseminator of information and yet is easy enough to use by the average person (Williams, 1999). The current attempt at the ideal solution is the online course management system.

Course management systems, as the name implies, are computer tools that provide an electronic means of organizing and managing a course's structure, material, communication, and participation. Examples of popular course management systems include Blackboard (Blackboard, 2006), WebCT (WebCT, 2006), and Moodle (Moodle 2006), the latter of which is open source software. Typically, these systems have attempted to incorporate the Internet's most popular communication tools, such as chat rooms, bulletin board services, email, voice over IP, and video conferencing into one easy to use, functionally cohesive interface. These systems typically have two viewing aspects. The first one is for use by the students, where they can, for example, download homework assignments or take quizzes, chat with other students, ask questions to the professor, view and/or participate in online lectures, or browse through the electronic BBS or secondary course materials for answers or discussions about issues of interest (Williams, 1999) (Petrides, 2000). In the last five to ten years, the use, development, and interest in such course management systems has grown significantly and as of 2003, online courses make up nearly three percent of higher education courses and seminar

courses (Badrul, 2003). While online course management systems have made great strides in improvement of the distance education experience, they all suffer from one major drawback: the instructor must structure the course information according to the (often rigid) layout and specifications of the course management tool. Besides being relatively inflexible, this requirement can place a barrier to entry into the online teaching world from a professor's point of view, especially if his/her existing course material will have to be totally reformatted to work with the course management system.

OfCourse, a course management system developed here at WPI, provides a new paradigm to course management systems in that it allows the tools to be embedded directly into the existing course material. This completely eliminates one of the most major caveats to the other course management systems mentioned previously. Professors wishing to use OfCourse need not totally abandon or reorganize their existing electronic material, which may have taken significant effort to develop in the first place. Instead, the professor simply keeps the existing structure of the electronic course information, which is usually in the form of web pages, as it is and simply drops in the relevant OfCourse tool, such as the chatroom, where and if appropriate. The OfCourse system was a fully functional software package at the beginning of this project, and had indeed been tested and used as a teaching aid by the previous IQP teams and by Professor Lemone herself for her online courses. It includes a chatroom, discussion board, file sharing utility, grade book, and administrative tools, among other features.

However, there were several aspects of the system that were either underdeveloped or lacking entirely. Certain key improvements were needed to take OfCourse to the next level in performance and ease of use from the viewpoint of both the instructor and the student. First and foremost, there was no way to perform a search across the myriad of very diverse and heterogeneous information that existed within the OfCourse system. A student or instructor wishing to find information relevant to a specific topic, keyword or subject, for instance, was required to manually browse through the multiple areas of OfCourse (such as the discussion board and file exchange) examine each piece of information that they found, and determine if it was relevant to what they were looking for. With the multitude of powerful computer-based search technologies that exist today, it was clear that this task was an unnecessary burden for the user. In

addition to the omission of an adequate search facility, each new course that an instructor wanted to teach required a separate installation of the entire system, including the application software as well as the databases. This made administration of multiple courses very tedious and cumbersome for the instructor, and provided no means for a student to access information contained in previous offerings of the same class, or to switch between classes. It also made version control and upgrading the software a major ordeal, because each of the multiple instances of the system had to be upgraded one at a time. Also, this looseness in the design introduced several major security holes that could allow even a novice cracker to gain unauthorized access to the system. Finally, there were various shortcomings in some of the more widely used subsections of OfCourse. For example, the human interface to the file upload tool was very unintuitive and difficult to use; the chatroom was annoying because it required the user's browser to continually refresh the page every few seconds in order to receive new chats, and the quiz system was loaded with bugs to the point of being virtually non functional. These issues underscored the fact that this otherwise powerful and robust distance-learning tool was in need of some major improvements.

The goal of this project was to provide these needed improvements, as well as explore other options to make OfCourse more useful still. This goal was fulfilled by realizing the following objectives: Reorganizing the software structure from a patchwork of codependent software modules into a cohesive multi-tiered plugin style architecture, finding major security holes within the system and fixing them, redesigning and/or replacing certain tools within the system to be more functional and user friendly, and conducting extensive research and development into the creation of a powerful and flexible discoverability search mechanism.

2. Background

An instructor needs to address several issues when deciding to teach a class online. One of the first tasks that must be preformed is to construct materials for the online classroom and curriculum by either creating new online content, or, more commonly, converting traditional media into electronic form. Then, an appropriate course management tool should be selected and deployed that acts as a powerful manager and disseminator of information, provides easy to use and effective communication tools, and a means to quickly and easily search for and find information contained within those tools. Each of these important topics in the administration of distance education is explored in this chapter.

2.1 - Considerations Regarding the Design of a Course Management System

Of all of the expert sources consulted during the research phase of this project, the single most agreed upon issue was the importance of the course management tools that are deployed to provide the proper functionality in a way the users of the system can easily understand (Williams, 1999) (Pallof & Pratt, 1999) (Khan, 2003). Indeed, it seems to be an unnecessary burden to place upon the students or instructors of any distance learning community to have to learn and master a convoluted, awkward piece of poorly written software on top of the demands of having to learn the actual course material. Such an experience could possibly disillusion an individual student or instructor on the idea of distance learning for a long time, possibly forever (Berge & Collins, 1995). On the other hand, an effective design can tear down the boundaries between different styles of learning while accelerating the rate of learning, providing the users of the system with a comforting, easy to understand, and seamless experience. This, in turn, will draw more and more people into the distance learning community (Kahn, 2003) (Berge & Collins, 1995).

2.1.1 – The Importance Of Application Familiarity

Most experts agree that the way to accomplish Application Familiarity is through the design and deployment of tools that are functionally the same or very similar to applications that the users of the system are already familiar with (Williams, 1999) (Pallof & Pratt, 1999) (Khan, 2003). In today's Internet age, the world is abound with various applications to do all number of different tasks. However, there are several pieces of software that have penetrated into the mainstream community and have become widely accepted. For example, most if not all people familiar with the Internet are at least marginally comfortable with applications such as email, bulletin board services, discussion boards or "blogs", instant messaging, search engines, and, of course, the world wide web, all of which can play a major role in disseminating information to online communities, such as virtual classrooms. Therefore, in the quest to design the optimal distance learning software, one has to examine the best and/or most popular applications for each of the aforementioned technologies and integrate their relevant functionality into the course management system. Then, when a student or instructor uses the system for the first time, they will already be familiar with what it is and how it works, especially if the interface closely resembles the interface to a program they already know how to use (Petrides, 2000) (Meyer, 2002). This can indeed go a long way toward encouraging instructors and students alike to considering distance learning.

2.1.2 – Electronic Mail

By far the most widely used form of communication across computer networks worldwide is electronic mail, or email. Almost everyone who uses the Internet has at least one email address and is familiar with how it works (Primary Research Group, 1997). It is therefore essential that any decent course management system be able to utilize email in an effective manner. For example, a directory of all students and instructors in the class should be available (unless a particular individual does not want to be contacted by email and wants their address to remain private). This way, students can easily form a electronic correspondence with other students or email the professor to ask a question or obtain more information. The professor can also use email to send important announcements to the entire class via a mailing list, since most people are likely to check

their email regularly, even if they do not happen to log onto the course management tool that day (Course Management Systems, 2006).

2.1.3 – Discussion Boards

Another Internet application that is extremely beneficial to distance education and has gained wide acceptance and popularity is the electronic discussion board. A discussion board can be implemented in its traditional way, using a central Bulletin Board Server (BBS) and clients that connect to it and download or upload content, or the traditional functionality of a BBS can be implemented by dynamic web pages over HTTP (Sherry, 1996). The latter method is usually called a discussion board, web log, or "blog" for short, but the functionality is essentially the same as a classical BBS.

These tools typically allow users to create topics of conversation, called threads, in which people can write their own replies to the conversation, called posts. Some systems apply an additional layer of abstraction and split up the threads into separate groups called forums, where each forum typically has a certain theme or subject associated with it. Features of the more useful discussion boards usually include a search feature, where users can search the text of threads or individual posts, and filter the results by a certain range of criteria, including how old the post is, the user(s) who wrote it, and how other users of the discussion board rated the post, if a rating system exists (Course Management Systems, 2006). Much information, including the survey results from the previous IQP, indicates that the discussion board is one of the most accepted and heavily used features of course management systems, including the myWPI instance of the Blackboard system at WPI (IQP, 2005). In fact, studies show that success in distance learning is directly linked to posting and to reading the posts of others. Posts ranging from lecture questions, discussion of homework problems to social and intellectual interaction between students and teachers can all easily take place through an appropriately designed discussion board (IQP, 2005). This underscores the importance of making the discussion board in OfCourse searchable, so that students can easily find information contained within this most widely used tool.

2.1.4 – Instant Messaging

Another widely accepted means of communication is instant messaging, such as chat rooms, which allow teachers and students to communicate with each other in real time. A chat room can also include an audio/visual information dissemination system, such as a web cam and videoconferencing system, enabling the professor to give lectures and discussions that the students can see and hear, as well as use to ask questions. This is far more powerful than just a basic text based chat room, because as most of the experts say that body language and tonality play a vital role in human communication (Williams, 1999) (Pallof & Pratt, 1999) (Khan, 2003) (Course Management Systems, 2006).

While these popular Internet applications are indeed a vital part of any well designed course management tool, an equally important aspect that the system should possess is an effective means of searching the data contained within each of these tools (Primary Research Group, 1997) (Meyer, 2002) (Course Management Systems, 2006).

2.2 – Search Tools

While the importance of the ease of the distance learning tools themselves cannot be stressed enough, an equally important issue is the means by which a student or instructor can coherently find and/or organize the information within the system once it has been placed there (Starr, 2000). There may be a tendency to want to group together related information, such as discussion board posts, files, chat logs and other sources of data that are related in some way. Indeed, the ease by which a user can do this plays a vitally important role in the overall distance education experience. It is somewhat analogous to a properly organized index in the back of a traditional textbook, in which the reader can look up a certain topic by keyword and quickly discover the page(s) that deal with or are relevant to that topic. This is a much more efficient process than flipping through the book page by page, hoping the eye will be fortunate enough to stumble upon the subject of interest while scanning each page one by one. The same is true of any repository of information; there must be an effective and efficient means of searching its contents.

2.2.1 – Discoverability Search

Discoverability search, developed by Thomas Watson and John Wiley in a joint effort between the computer science departments at MIT and the University Of California At Berkley, is a method of searching for keywords, phrases and questions across a multitude of heterogeneous and loosely connected data sources (Watson & Wiley, 2001). It has been successfully deployed in a wide range of searching applications, such as libraries, corporate data warehouses, medical information repositories, expert systems, government and telecommunications databases, and the World Wide Web itself (Watson, 2005). The general concept is to subdivide data into objects. These objects are analogous to objects in object oriented programming languages, such as those of the fourth and fifth generation. Each object definition is a template, consisting of data fields and functions, while each instance contains the information that belongs in each of the fields as well as references (pointers) to the functions contained in its template. The functions are then called to access and/or manipulate the information contained in the object instances. Discoverability search is in essence a subset of the functionality of OOP in that the data fields are limited to information called metadata, and the functions are limited to search routines. These two vital concepts are discussed in the following sections.

2.2.2 - Metadata

Metadata is essentially "data about data" (Watson & Wiley, 2001). It is information that describes, or gives meaning to, other information. A popular example is that of a zip code. By itself, "01720" is totally meaningless, but when the metadata "Zip Code" is assigned to it, it then becomes useful (Watson, 2005). While a book may have the metadata of title, author, year, subject, publisher and so on, a car may have the metadata of make, model, year, color, horsepower, transmission, etc. This is how instances of very different entities can be brought together into a single unified search space. Using the above example, a search on "year=2001" would divulge all of the books published and cars manufactured in that year that was known to the system. However, if

the search was performed on "subject=computer science", only books would be returned, as results containing cars would obviously be meaningless (Watson & Wiley, 2001). Using the technique of discoverability search, relevant metadata is assigned to each category of information that the developer wishes to make searchable. Watson and Wiley have described over five hundred different categories of metadata (Watson & Wiley, 2001). It then becomes the task of the developer to select from these the ones that are relevant to the particular category of data, which is to be searched.

2.2.3 - Metadata Spaces and Discoverability Vectors

While choosing appropriate metadata is indeed a critical aspect of the design, an equally important task is to develop the functions that determine how relevant a given query string is to the metadata. Watson and Wiley have described the concept of a discoverability vector, which is what each of these functions is expected to return (Watson & Wiley, 2001). Essentially, a discoverability vector is a vector, extending from the origin, within a metadata space (Watson & Wiley, 2001). A metadata space is an orthogonal, multidimensional space where each axis is of unit length and represents how well the search string matched one particular piece of metadata (Watson & Wiley, 2001) (Watson, 2005). The dimensionality of the space is exactly equal to the number of metadata fields associated with the object being searched (Watson & Wiley, 2001). Each component of the discoverability vector, therefore, is a number between zero and one, inclusive, that indicates how relevant the search string was to the one piece of metadata represented by one of the axes (Watson & Wiley, 2001). For example, if the object being searched had two pieces of metadata associated with it, the discoverability vector would be two dimensional, with the "x" component indicating how well the search string matched the first piece of metadata, and the "y" component indicating how well the second piece of metadata was matched. This is illustrated in figure 2.1. A perfect match on both pieces of metadata would yield the vector (1,1), a partial match might result in (0.4,0.7), and no relevancy at all to either piece of metadata would produce (0,0). It is in this way that the results from the search can be sorted by relevancy. The greater the

magnitude of the vector, and the closer the angle it makes with each of the axes is to pi over four (45 degrees), the more relevant that particular search result is.

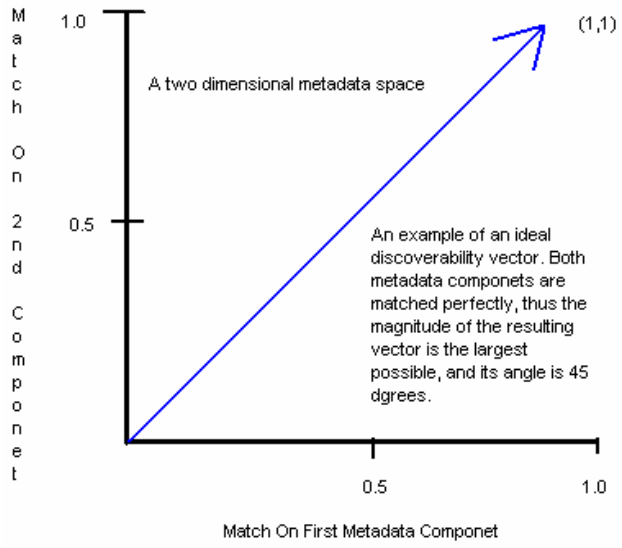

Figure 1 - Example Of A Discoverability Vector

2.2.4 – Discoverability Search and OfCourse

In the design chapter that follows I will detail how the idea of discoverability search was used to design OfCourse's search API. This made each of OfCourse's plug-ins searchable from a top-level interface and thus providing a common method of accessing the vastly differing types of information contained within each.

2.3 - Conclusion

This chapter explored some major issues in deploying an online classroom and building an effective distance-learning environment. Course content must be integrated with a properly designed, and easily searchable, course management system, such as OfCourse, in order to be disseminated over the Internet. In the next chapter, Design, I

will examine the technical details of how we applied this background research in distance learning toward the project goal of improving OfCourse.

3. Design and Implementation

The goal of this project was to refine OfCourse into being a modular, secure, easy to use and searchable online distance-learning tool that is relatively easy to install and maintain by a professor who is at least somewhat familiar with the Internet and World Wide Web. In the last section I focused on the background and theoretical issues related to the design and use of such a course management system. Now I shall examine the specific design issues used to redesign OfCourse. I will discuss the design of the specific updates, including the design of the new plugin architecture, redesign and modularization of the myriad existing components of the system, and the design of the new discoverability search component, as consistent with the stated goals from the Introduction Chapter.

3.1 - Design of the Plugin API Architecture

In software design, the development of plugin architectures consists of dividing the system into two distinct parts: The upper level Application Program Interface, or APIs, and the plugins themselves. What separates the two is a very simple concept, the upper level APIs provide the *general* functionality that is common to all or most of the entire system, while the plugins supply functionality that is *specific* only to a subset of the system. The plugins can be thought of as the additional features, whereas the upper level APIs usually encapsulate much of the core functionality. Many examples of plugin architectures abound in Computer Science, from microkernel operating systems to web browsers. The goal of the project was to rearrange OfCourse into a plugin style, modularized architecture where the functions that are common to all or most of the pages in OfCourse (such as logging in, connecting to the database, etc) exist in the high level APIs, while the individual features of OfCourse (such as the chat room, discussion board, etc) are organized into the plugins. In addition to making the code itself much easier to develop and maintain, the plugin approach also allows much easier development and installation of any future add-ons or updates to OfCourse, since they can be implemented

as plugins and then "plugged into" the existing, well defined high level APIs. While providing these unique benefits to the programmer, this architectural change had desirable effects for the user as well. For example, a professor wishing to teach a course online only needs to install and configure the features (plugins) of OfCourse that he or she wishes to use, since plugins operate independently of each other. This has the obvious benefit of an increase in both flexibility and robustness from the user's point of view. I will now examine the methodology behind the development of each of the high level APIs and plugins, as well as the reasons why they are appropriate.

3.1.1 – Design and Implementation of the Login API

I designed the login application program interface to be responsible for authenticating a user, then recognizing and remembering that particular user when he or she returns to any page in OfCourse. Since checking if and who is logged in when a page request occurs to any page, this functionality was an ideal candidate for a top level API. This API contains the login page itself, which I designed to look and feel exactly like the existing login page. Once a user supplies valid credentials, users need not log in again from that computer, since the module issues a long duration cookie to the web browser. Then, the API adds the user-userid tuple (contained in the cookie) to a list of authenticated users that it maintains. This way, when a page residing in one of the plugins needs to check if the user making the request is logged in (or if he or she has administrator privileges), all it must do is call a function in the login API. If the user is logged in, the function simply returns an appropriate value. However, if the user is not logged in (supplies no cookie or an invalid cookie) I designed the function to redirect the user to the login page. Once the user successfully logs in, he or she is then redirected back to the page that was initially requested. Note that this represents no fundamental change to the original behavior of OfCourse, only a fundamental change to the architecture.

3.1.2 - Design of the Database API

In order to have persistence between page requests of state and data within OfCourse, a plugin must somehow connect to and manipulate information within a database. The original version used SQL queries to a MySQL database. While not a bad choice in and of itself, the application was heavily dependant upon the Mysql database driver, which has been obsolete for about the last seven years. In addition to this minor drawback, the extensive use of the Mysql driver throughout the OfCourse system made the application highly platform dependant, since no other database server other than MySQL would work. My design of the new database API not only provides a common connection point for all of OfCourse's many plugins and APIs, but also includes the updated version of the Mysql driver, known as DBI (Database Interface). In addition to being the preferred modern method of connecting to a database from a web interface (or any other application), the DBI database driver, unlike the Mysql driver, is truly independent of the type of database server being connected to. This allows OfCourse to be installed on a web host regardless of whether the host provides MySQL, mSQL, Microsoft SQL Server, Access, etc as its database server. Thus, the OfCourse database API is designed not only to provide a common entry point for plugins to access and store information in the database; it also decouples the hard-coded dependency between OfCourse and MySQL.

3.1.3 - Design of the Admin API

The admin API is responsible for providing several much needed functions to the general architecture of OfCourse. Its primary purpose is to encapsulate the administrative functions, such as approving or disapproving a new user, maintaining the class directory list, and administration and maintenance of user accounts. This API includes the current approve/disapprove user's page and the class list. The lurker user tracking facility is also integrated into this API, and works in close connection to the Login API for a detailed tracking routine of login/logout times and user activity. The look and feel parameters of the system can also be administered here. This provides much greater flexibility in the

configuration of the UI, because one change here takes effect everywhere throughout the entire system. All in all, the admin API is a much needed and valuable improvement. It serves as a central location for administrative tools and plugins to configure and administer users and user accounts, grades, look and feel, and make any changes to the current installation of OfCourse.

3.2 - Security Considerations in the Design Process

Any system that depends on the open Internet to operate is vulnerable to attack by malicious hackers, sometimes referred to as crackers or cyber criminals. This vulnerability must be taken very seriously with regards to online classrooms. For example, the academic honesty policy here at WPI is put in place to ensure that any student who is granted credit for a class is given the credit because he or she has gained "mastery of the material", and not because they cheated, plagiarized, or in some other way tricked the professor into believing that they mastered the material when in fact they hadn’t. This possibility of academic dishonesty takes on a whole new dimension when the "classroom" is essentially a computer application running on an open wide area network like the Internet.

3.2.1 – Cookie Generation Security Hole

One of OfCourse's strongest assets is its open nature. Only one login is required, and then the application remembers the computer's IP and browser, which authenticated successfully. This eliminates the need to login each time someone needs to access the course material. A cracker could, however, gain access and steal one or more accounts if he/she were to spoof his IP address and snoop the cookie value from a legitimate user, both of which are not hard to do for most programmers. The redesign to plug this hole was a relatively easy one, however. I simply designed a new "rolling code" id system that is now used for generating cookies. In this way, a new cookie id, which is totally independent of the old cookie id, is produced for each new request by the legitimate user making such an attack as mentioned above nearly impossible.

3.2.2 – Login System Security Hole

A much more vulnerable part of the system was the login page itself. It ran over open plaintext HTTP (Hypertext Transfer Protocol) and did not use SSL (Secure Socket Layer). Therefore, whenever users logged in, including professors, their username and password were sent over the open Internet in clear text. It does not take a significant amount of skill to snoop an open network and reap (collect) passwords. It would compromise the safety and integrity of the entire class if this were to happen, especially if the account that was stolen had instructor privileges. The cracker could easily change grades, destroy course material, perform a denial of service attack, or any other nasty tricks. Clearly, this potential situation would prove disastrous for not only OfCourse, but the entire online educational community's reputation would be undermined as well. Since this was obviously an unacceptable situation, I designed in the necessary security fixes (use of SSL for logins) while still maintaining the unique and convenient "log in once" feature of OfCourse.

3.2.3 – User ID Security Hole

I discovered another major security hole during the design process and that was the manner in which the user ids were generated. The user id is supposed to be a random, unique string that is assigned to each user of the system, professors and students alike, when their account is approved. This id in turn is then sent to the user's web browser encapsulated within a cookie. Then the web server can determine which user is issuing a request to the OfCourse system. The problem was that the user management code was written to generate user id's by performing an md5 hash on an auto-generated primary key value used to uniquely identify the rows in the user table within the database. However, the way MySQL generates these values is that it starts with the value of "1", and for each new row added to the table, the value is incremented by one. So, for example, the first row in the user table (which would be the entry for the administrator, since she was the first user added to the table), would have a user id which was the md5

hash of "1", the second user added (perhaps a student, perhaps a TA) would get a user id of the md5 hash of "2", and so on. Anyone who knew of this behavior, or figured it out by observation and/or experimentation, could easily crack into the administrator account by simply sending the web server a cookie which contained the md5 hash of "1". This, of course, was a major security hole because with admin privileges a cracker could wreak all kinds of havoc to the system including, but not limited to, changing his or her grades, or the grades of other students, in the grade book. I therefore made the decision to redesign the user management code to generate md5 hashes for users based on a combination of their username, password, Unix timestamp of when the account was created, and a random number.

3.3 - Design of the Individual Plugins

I redesigned several of OfCourse's plugins to be more functional and user friendly. Others needed to be replaced entirely, because they were either not working or were very difficult for new users to figure out.

3.3.1 - Design of the Chatroom Plugin

Chat rooms are not for just text anymore. In fact, there are several features that can be incorporated into traditional chat room programs and interfaces that both enhance and sometimes substitute for the real time text messaging experience. These include voice over IP features that allow the students and teachers to send two-way voice messages to each other and, in some cases, even emulate the traditional full-duplex voice mode of a telephone. Also, if high-speed Internet access is available, video conferencing can take place through a chat room, allowing the professor to provide lectures and/or office hours in real time with the students able to hear and see them on their screen. Electronic whiteboards can even be incorporated allowing the professor to draw diagrams, equations or notes right on the screen. This provides the students participating in the "chat" to see these drawings in real time. With the release of the new tablet PC interface, where the user uses a pencil-like device to draw directly on the screen, such a real time electronic white board feature shows promise for increased use in the future. While voice and video chat are somewhat bandwidth intensive, electronic whiteboard

applications can actually be designed to use small amounts of bandwidth, since only changes in the image (which occur at a relatively slow speed) have to be sent over the network.

As mentioned in the Background Chapter, the programmer needs to look to popular chat clients and emulate the interface when designing a chat room feature for a course management system, thereby reducing the time it will take the average computer user to learn how to use it. Popular chat room software applications such as ICQ, IRC, and AIM were therefore appropriate candidates to look to for ideas on how the new chat room plugin should look and feel. Taking these considerations into mind, Seth Hunter, a Graduate Student who was also working with Professor Lemone to improve OfCourse, built in a video conferencing feature based on the popular TinCam software. This, as well as an electronic white board feature, greatly improved and standardized the user interface to provide students and teachers alike with a very useful and very powerful chat room application.

3.3.2 - Design of the File Sharing Plugin

In keeping with the central theme that tools designed for online learning should mimic applications that people are already familiar with as much is possible, it makes sense to use a file sharing tool that closely resembles the way traditional computer file systems work. Even among casual computer users, the notion of directories (folders) and files is pretty much common knowledge. Therefore I decided that the file upload and sharing tools of OfCourse should implement directories and files in a similar manner to a basic file system. An invaluable asset a course management tool must possess is the ability to quickly and easily distribute and locate relevant course documents and files.

Due to the limited amount of time that was available for the development phase of this project, I asked for and obtained permission to use an open source third party file-sharing tool instead of constructing one from scratch. I located several good choices, mostly from sourceforge.com and other open source websites. Ease of use, and the ability to quickly and easily modify the application to fit into the plugin structure of OfCourse were the main deciding factors used to select the appropriate application. I downloaded

and installed each of three final candidates and set them up as stand alone applications for Professor Lemone and myself to "play with" to see which one would be the best fit. After this short experimentation phase, we agreed that the best choice was a file upload application called Promia (Promia, 2006). I subsequently re-designed this software package to be a plugin to the larger OfCourse application.

3.3.3 - Design of the Quiz System Plugin

While there were two quiz systems that were originally designed for this project, neither choice was adequate. The first quiz system included with OfCourse was poorly designed and implemented, badly broken and so loaded with bugs that it would have probably taken less time to write a new one than to fix the old one. So, for these reasons, Professor Lemone and I decided to scrap this piece of code. The second quiz system, while very well made and highly functional, was designed to be a stand-alone application. I considered modifying it to fit into the new plugin design, but documentation for the code was non-existent (most of the source code did not even contain meaningful comments), and the developers of the code had either left WPI and/or were very difficult if not impossible to contact. Once again, the decision was made to not use this quiz system based on the amount of time it would take to reverse engineer a relatively large and complex piece of code and then modify it to accept the new plugin architecture. A new quiz system was therefore needed, since the two available quiz systems were unacceptable.

Once again, I looked to open source software from sourceforge.com to find an acceptable, available third party tool rather than build an entirely new quiz system from scratch. However, unlike with the file sharing tools, there were much fewer choices available. At first I noticed all of the available choices were applications that were originally designed as web based polling and voting systems. These would need to be modified to not only fit into OfCourse as a plugin, but to include points and grading as well. Then, a fully functional quiz system called PAM was found, which was not from sourceforge but from its own independent website (PAM, 2006). It was very user friendly, included extensive and detailed documentation of the code, and was completely

free and open source. When I examined the documentation, it became clear that PAM would be quite easy to integrate into OfCourse as a plugin. Therefore, I selected PAM and redesigned it to be OfCourse's new quiz system.

3.4 – Design Of The Discoverability Search Tool

With the enormous popularity and usefulness of Internet search engines such as Google, Yahoo and Lycos, just to name a few, it should come as no surprise that any decent course management system should incorporate a powerful search facility to quickly and easily find desired information within the system. This feature is the one feature discussed so far that is noticeably lacking from OfCourse, and adding it was a major focus of this project. With information stored in so many forms, including discussion board posts, online chat sessions, course documents and materials, files, and lecture notes, Professor Lemone and I believed that by adding some method of unifying the information through a common search procedure would be doing OfCourse a great service.

OfCourse contains many different forms of popular Internet applications, such as a discussion board, chat room, file exchange, user directory, and HTML pages. Each one of these modules, or applications, stores vastly differing types of information, and manages the information in highly differing ways. While this plethora of useful tools is no doubt a very good thing for a distance learning application to have, it made the task of constructing a searching system that could effectively retrieve information from this wide variety of data far from trivial. A general, all-purpose solution was not acceptable because a system that would work well for searching a discussion board, for instance, might not be the best way to search web pages. Indeed, a solution was needed that would allow the programmer to custom tailor an appropriate search algorithm for all of the different plugins, while still providing the user access to a unified, cohesive top level search interface. Fortunately, as described in the Background Chapter, research revealed an existing technique to do just that called Discoverability Search.

3.4.1 - Design of the Search API

My primary focus in the design of the search API was to encapsulate the functionality that was to be used by some or all of the individual search modules. This was, in essence, the function that dealt with the computation of the metadata space and the discoverability vectors. Also, the search interface itself was to be contained here, since this would provide the common point of entry for the user to initiate a search.

3.4.2 – Constructing the Discoverability Vectors

I designed functions to take metadata, in a textual form, and compare them against a search string and then return the relevancy as a number between 0 and 1 inclusive. There were two techniques that were considered appropriate for doing this during the design process. The first took a binary approach to the discoverability computations of the search string to the metadata. If the search string appeared in the metadata itself, the function returned a 1, otherwise, 0 was returned. This was appropriate for use with metadata that was short in length, such as the title of a discussion board post. The second type of function I designed to take a list of metadata strings and count the number of occurrences of the search string within each piece of metadata. The metadata string which contained the most occurrences was to receive a value of 1, while each of the others that contained some number of occurrences less than the most but greater than zero was to receive some number less than one but greater than zero. Finally, metadata strings that contained no occurrences of the search string were to naturally receive a zero. For example, if the max number of occurrences was five, and another piece of metadata contained three occurrences of the search string, this piece of metadata is assigned a value of three over five, or 0.6. This is consistent with one of the techniques for handling this situation described by Watson and Wiley (Watson & Wiley, 2001). Since it was known at the design stage that an implementation of this procedure would be slower in run time than the binary method, this was reserved for metadata that was longer in length or more complex in structure. For example, the content of a discussion board post would fit this category. I decided that such content demanded a finer grain of discoverability measurement than a simple yes or no, one or zero answer given by the binary method.

The values returned by these two discoverability-measuring functions were used to construct the discoverability vectors themselves, one component at a time.

3.4.3 – Analyzing the Discoverability Vectors

Once the discoverability vectors were constructed, their length and their angle relative to the axes of the metadata space had to be calculated. I made the design choice to have a third function handle this. This is where weighting of the individual axes could be brought into play, if necessary. For example, if the designer of the discussion board search module decided that occurrences of the search string in the title were more important than occurrences within the content, the title metadata component could be weighted more heavily than the content component. Thus, I designed two functions for computing the length of the discoverability vectors, one that applied weighting, and one that did not. I further realized that the non-weighting function was just a special case of the weighting function where the weights of all the axes were equal. Once a complete discoverability vector was created and its length and angle measured, it was reduced to the two-tuple, or couple (r, theta). It was then only necessary to sort the discoverability vectors in non-ascending order.

3.4.4 – Sorting the Discoverability Vectors

For the purposes of sorting, I used a special implementation of Randomized Quicksort. Instead of simply sorting numbers, as the classic implementation of Quicksort does, an appropriate method had to be designed to take both the length and the angles into account. Once again, there were several design choices available (Watson & Wiley, 2001). Since the size of the search space (quantity of data being searched) was small comparatively (as opposed to an entire library or a cooperate data warehouse, say), the procedure that I selected was the simplest non-naive procedure recommended by Watson and Wiley (Watson & Wiley, 2001). The idea is to basically sort on the length of the vector, while using the angle as a tiebreaker if necessary. Therefore, I implemented a suitable version of Quicksort to accomplish this.

3.4.5 - Designing the Search Modules for the Individual Plugins

While the design of the search API focused on how to construct and manipulate the metadata space and the discoverability vectors on a global or abstract level, the design of the search modules for the individual plugins needed to be focused on construction of the search space in the form of the object tree, and to search on the appropriate pieces of metadata.

The object tree consists of the individual objects to be searched as well as their parent-child relationships with each other. For example, discussion board threads represented as objects are a natural selection for the parents to discussion board posts, since threads contain posts. While it is possible in theory to construct object inheritance trees that is infinitely deep, in practice a line has to be drawn somewhere that limits how fine grained the object representation will be. This is a similar situation to designing object trees in object oriented programming languages, eventually a point is reached where objects are formed that have no children. These "leaf" objects represent the frontier of the tree where the highest level of desired specificity has been achieved. Once again, the time constraints of the project were the major limiting factor in the design as to how deep the object tree should be constructed. After careful consideration, Professor Lemone and I decided that the depth of the object tree should be limited to no more than two or three levels, to prevent the complexity of the object representation from getting out of hand.

Having only two or three levels to work with, I had do give careful consideration to the choice of each object and how it fits into the tree to maximize the effectiveness of the representation while minimizing the complexity. For the discussion board, I made topic objects and designed them to be parents to thread objects. Similarly, thread objects were made the parents of post objects, which in turn were the leaves of the tree. Notice, for example, how the tree could have been extended deeper by making, say, content objects children of post objects, sentence objects children of content objects, and word objects children of sentence objects. However, with the design choice to limit the depth to no more than three levels, I stopped at posts. In a similar way, design choices such as

these had to be made for all of the other plugins as well. The file sharing utility had two levels, consisting of directory objects being parents to file objects. With the chat room, only one level was necessary: the chat archives themselves were designed to be the only object. The same one level deep strategy was used for the user directory, where just the users themselves were the only objects necessary. Finally, for the static web content, I made the HTML pages themselves to be parent objects of the content of the HTML pages. This design, I believe, provided a simple yet highly robust and usable object tree that could be easily extended by a future MQP team if the need arises.

Once the design of the object tree was complete, the appropriate metadata needed to be selected to serve as the data members to these objects. Again, time constraints played a factor in the design considerations here. The choice of metadata to use for the search space had to be large enough to be effective and useful, yet small enough to realistically implement within the allotted time. While Watson and Wiley have defined over five hundred different categories of metadata, it was obvious that a very small subset of these would need to be used in the design of OfCourse's discoverability search tool (Watson & Wiley, 2001). Basically, the choices were restricted to textual information that had the highest chance of providing a high level of selectivity between each object. Table 3.1 below summarizes the choices of metadata that were made for each object within each module (or plugin) within the system, according to the structure of the object tree discussed previously.

Table 1 - Metadata Used For The Various Plugins

Discussion board	**Object Name**	**Metadata Fields**
	Topics	Name Of The Topic
	Threads	Title Of The Thread, User Who Created The Thread
	Posts	Title Of The Post, Content Of The Post, User Who Created The Post
File Sharing Tool	**Object Name**	**Metadata Fields**
	Directories	User Who Owns The Directory
	Files	Name Of The File, Description Of The File, Content Of The File (if file is textual)

Chat Room	**Object Name**	**Metadata Fields**
	Chat Archives	Title Of Chat Archive, Content Of Chat Archive
User Directory	**Object Name**	**Metadata Fields**
	Users	First Name, Last Name, Username, Email Address
Calendar	**Object Name**	**Metadata Fields**
	Events	Date, Event Name, Event Description
Static Course Content	**Object Name**	**Metadata Fields**
	Web Pages	Title, Keywords
	Web Content	Text Within Web Page

3.4 - Summary

In this chapter, I described how this project redesigned the structure of OfCourse from a loose-knit, heterogeneous collection of application tools into a cohesive, efficient plugin style architecture and provided much needed security fixes to prevent crackers from gaining unauthorized access to the system. Also, I described the selection and modification of a chat room, file-sharing tool and quiz system. Finally, I detailed the design process involved in the development of a new discoverability search tool and object-metadata semantic structure. In the next chapter, Results, I will discuss the results related to these designs, as well as present the testing methodologies that were used and results of the tests that were applied during the course of this project.

4. Results

This chapter explains the results obtained from the implementation of the research, redesign and improvements to OfCourse described in the previous chapters. I will discuss the specific impacts on the software, as well as the testing methods I used and test results I obtained. First, I will describe the results of the implementation of the new top level APIs that facilitate application wide communication between the various plugins and enable multiple classes to be administered from a single installation. Next, I will discuss the results from the redesign and/or replacement of the individual plugins such as the file exchange and quiz systems affected by these changes. Then, I will describe the results of the development of the new discoverability search tool. Finally, I discuss the tests that were performed on the system and the fixes and changes that were made as a result of these tests.

4.1 - Results of the Implementation of the APIs

The most tedious aspect of the implementation phase of this project consisted of developing the system-wide Application Program Interfaces, or APIs, to unify all of the various tools contained within OfCourse. The original code base was very heterogeneous in nature because so many developers with varying levels of competence and expertise worked on it over a wide range of time. Because of this it was challenging to get all of these disparate pieces of code to communicate with each other through a common interface. While time consuming and tedious by nature, however, I realized this task was vital to the ongoing success of OfCourse because it enabled the application to not only be easily extended by future developers, if necessary, but it enabled a single install of the OfCourse system to handle (theoretically) an infinite number of classes. Despite the obvious benefits to the instructor in not having to perform a new complete installation every time she wanted to spawn a new course, this important improvement made the application much more secure, efficient, and easy to upgrade. I now discuss the results of these implementations of the new API structure in the following subsections.

4.1.1 - Results of the Implementation of the Database API

In the past, OfCourse used a single database to store all of the information or state of one single course (see figure 4.1). While this design was perfectly acceptable in and of itself, the shortcoming was that the application had no way of switching between databases to handle multiple courses. This, for the reasons described previously, was unacceptable and needed to be fixed.

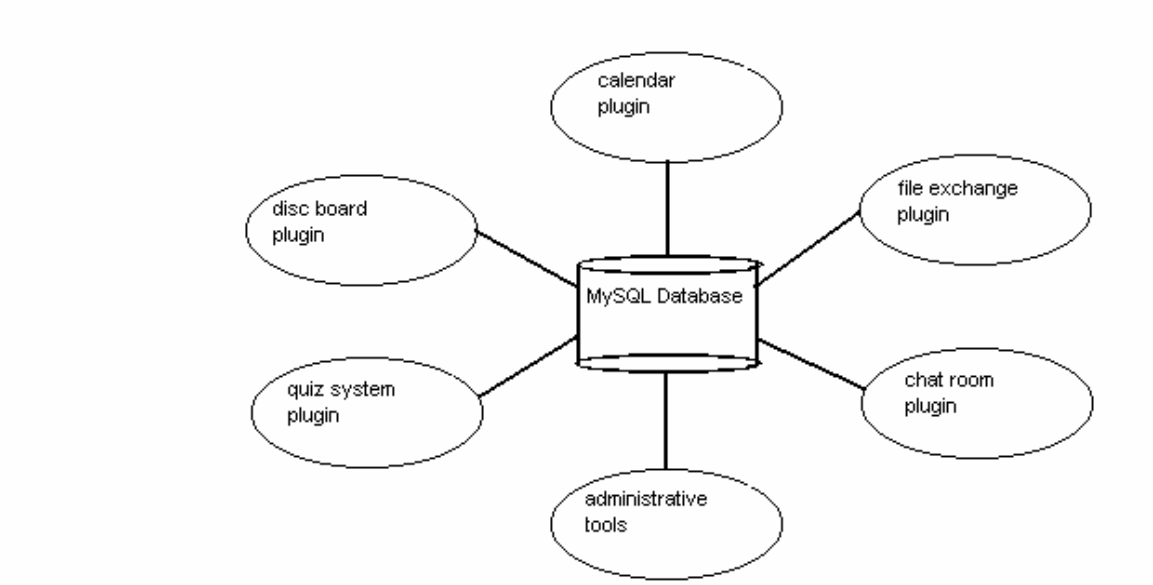

Figure 2 - Previous OfCourse Database Connection Handling

Two possibilities existed to overcoming this discrepancy. The first entailed redesigning the schema of the database, adding a new column called "course_id" to each of the existing tables. This "course_id" field would be given a different value for each new course. Originally, this is how I set out to accomplish the one-install-many-courses theme. However, early in the implementation phase, I realized each of the existing database queries (SQL statements) needed to be altered to include this new "course_id" field, and many would have to be totally rewritten to perform inner joins on this value as

well. While not difficult in and of itself, the problem was that the original developers of OfCourse decided not to separate the database interface from the user interface, as is usually done in deigning web-based database applications. Because of this, the SQL queries were loosely scattered throughout the entire application, often appearing right inline with procedures that generated the actual HTML that formed the web pages. This required that each and every code file within the entire application would have to be thoroughly examined to locate the relevant database queries and replace them with suitable SQL statements that included the “course_id” field. Considering the application consisted of over a hundred different Perl and PHP files, I therefore decided that this method would be too time consuming to accomplish within the six-week implementation period. It was then that I realized that the best way to implement the needed change was to use the second possibility, which was to use a new database for each new course.

To use a different database for each new course required only that the functions responsible for *connecting* to the database be made aware of the course in which the user was participating. To accomplish this, I added a second persistent HTTP cookie to the interaction between the client and the server. In this way, when a user logs in, he or she selects the course they wish to participate in, enters the username and password for that specific course, and clicks the login button. The system, after verifying the login credentials are acceptable, sends back to the user's browser two cookies: One containing the user's id as it appears in the database of that specific class as well as a cookie containing the course id of that class. Now, all further requests to the web server include cookies containing both the course_id and the user_id. The database API then uses the course_id to decide which course database to connect to, while the original OfCourse code can carry on as usual. I made this happen by hooking all database connects from the original OfCourse code through the new course_id-aware database API. This proved to be far less time consuming, since only the functions responsible for *connecting* to the database needed to be rewritten and moved into the API, rather than rewriting each of the numerous functions that *queried* the database. The result is a seamless transition from one installation of OfCourse being able to handle only one course, to the new paradigm of one installation being able to handle as many different courses that the instructor wishes

to teach. Figure 4.1 illustrates how this is done. Note that each tool (plugin) in OfCourse is made to make calls to the database API to get a database connection.

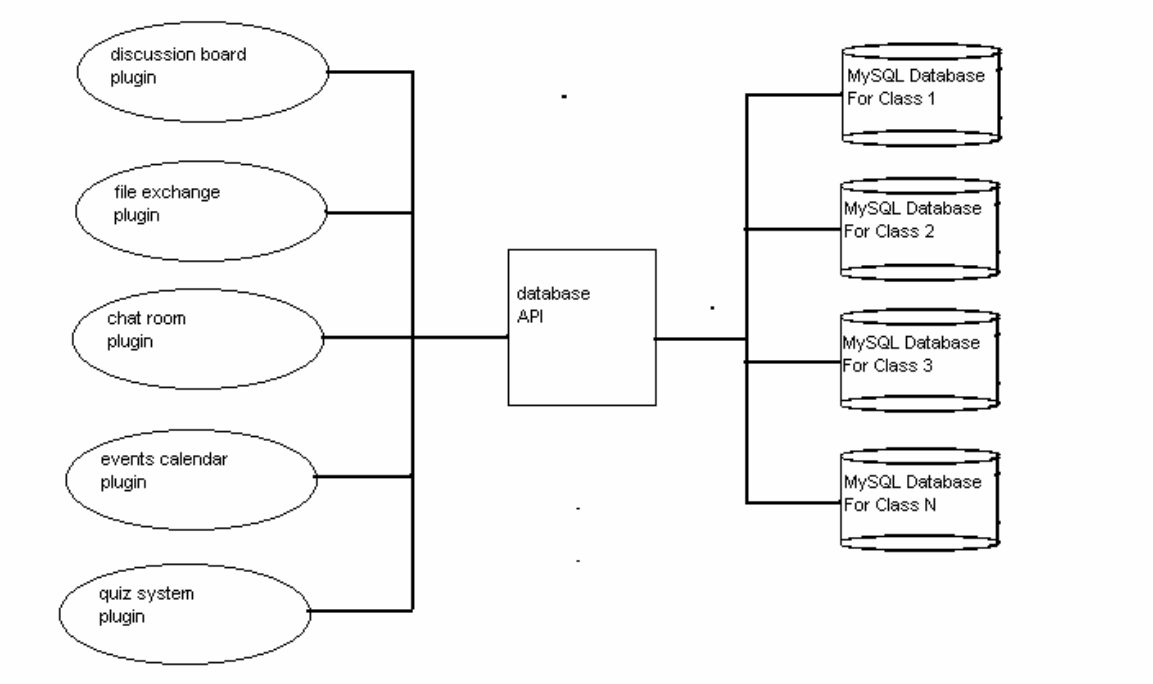

Figure 3 - New OfCourse Database Connection Handling

4.1.2 - Results of the Login API

The results of my implementation of the login API vastly improved security and user management. First, I redesigned the login page itself to include a drop down menu to select the course the user wishes to log into. This menu is then used to set the course_id cookie described above, which the database API uses to connect to the appropriate course database. In addition, it allows a user who is taking more than one course (or who wishes to access information in previous offerings of the same course) to easily switch between classes. Then all that is necessary is to select the desired course from the drop down menu; the username and password only need to be entered the first time the course is accessed. Next, I added code to the Login API to authenticate a user, determine their privilege level (instructor or student),

and to manage the cookies in general. This code is primarily used during the login or course-switching process. By determining the privilege level, through the user_id cookie, OfCourse's plugins determine whether or not to display administrative options to the user. Finally, I re-implemented the entire login process over Secure Socket Layer, or SSL, so that passwords sent over the network were in encrypted form rather than in plain text. This change eliminated one of the largest security holes that previously existed. The result is an easy to use, robust and secure login page that will be highly intuitive to users who have even the most minimal exposure to web-based applications. Figure 4.2 shows a screenshot of the login page that the user sees when logging into OfCourse for the first time.

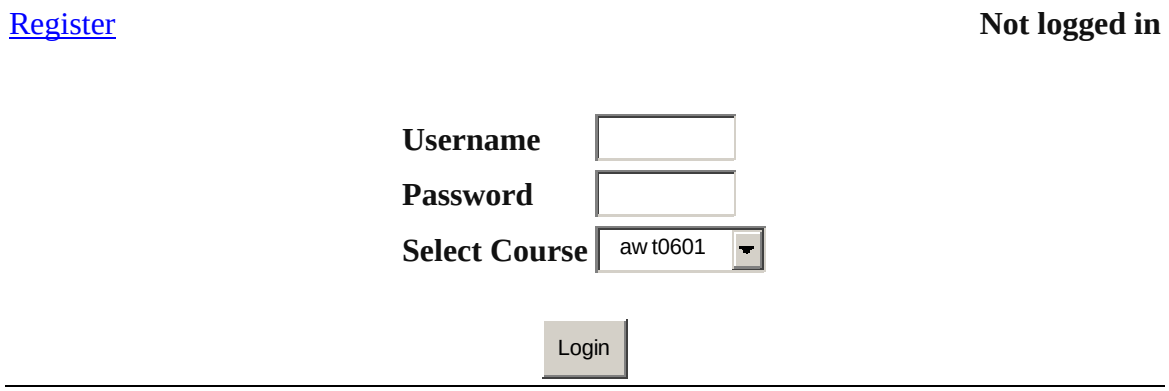

Figure 4 - Screenshot Of The Login Page

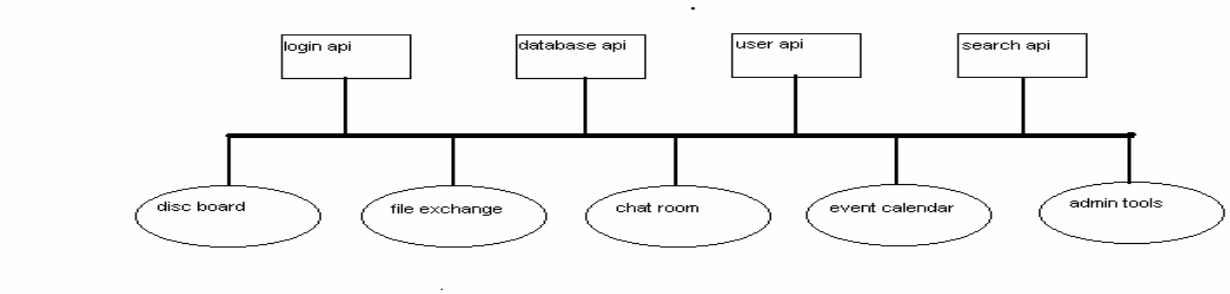

Figure 5 - Diagram Of The New Plugin Style Architecture

4.2 - Results of the Implementation of the Plugin Architecture

Once the APIs were properly implemented and tested, I began adapting the individual tools to the plugin architecture. I accomplished this by hooking certain key functions within the *functions.cgi* script. This script was originally responsible for holding commonly used functions across the various plugins. For example, this module contained functions responsible for database connectivity, authentication, user management, and cookie management. With the attention to detail given in the construction of the login, user, and database APIs described earlier, adapting the existing tools to be compatible with the new plugin architecture was relatively easy. It required only replacing the bodies of these functions with appropriate calls to the relevant API functions. Although this does introduce a slight amount of inefficiency, in that two function calls are required by a plugin instead of just one (one call to the original function in functions.cgi, then another call to the replacement function in one or more of the APIs) I decided that this was a better approach than to try to replace the function calls with API calls in each of the over one hundred different scripts. The time required to do so was much less, and the probability of introducing bugs was significantly lower as well. Therefore, I made the decision to sacrifice a modicum of performance for the sake of decreased development time in the form of finding and changing every single function call to *functions.cgi* in each of the plugin scripts, higher reliability, and less time

debugging. I used the time that was saved to work on other areas of OfCourse that were more important to its overall improvement, such as replacing the file exchange, chat room, and quiz system with newer, better versions.

4.2.1 - Results of the File Sharing Plugin

The first plugin I replaced was the file sharing system. As I mentioned in the design chapter, the original file exchange tool was unintuitive and difficult for most users to understand how to use. After successfully locating and testing the Promia web based file sharing system, I realized what changes the program needed to make it compatible with the new API architecture of OfCourse. It was the code that handled users and database connectivity.

In the standalone version of Promia, the application maintained its own user accounts and featured its own login and authentication scheme. Clearly this needed to be changed, because requiring the user to login to the file sharing application after already logging into OfCourse was just ridiculous. Therefore, I altered the code that was responsible for handling users within Promia to make calls to the user API within OfCourse instead. This was somewhat tedious, as Promia and OfCourse handled users in fundamentally different ways. For example, Promia had no notion of separate user authority levels, such as administrator and user; it simply treated all users as having the same privileges. Therefore, much of the code that I produced acted as a translator between the two different user management paradigms.

Also, Promia had its own database connectivity and database management modules, which needed to be modified to properly communicate with OfCourse's database API. This was less of a challenge than modifying the user management code, however. I simply replaced the calls to connect to the database with appropriate calls to the functions in the database API. The database management system itself then required only trivial modifications to be compatible. I produced a working version of modified Promia, ready to be plugged into OfCourse. For more information on the specific details regarding the integration of Promia, see Appendix C.

jwf2's Filespace

Upload A File

Your Files

Filename	Adjust Sharing Parameters	Delete
test	Adjust Sharing Parameters	Delete

Files Shared With You

Filename	File Owner	Remove Share
Prof Lemones File	Klemone	Remove Share

Figure 6 - Screenshot Of The File Exchange Utility

4.2.2 - Results of the Quiz System Plugin

As I explained in the previous chapter, the two existing quiz systems that originally came with OfCourse were unacceptable. It was therefore necessary to integrate the new quiz system, PAM, into the OfCourse plugin architecture. I did this in the final days of the implementation phase of this project, after the completion of the much more time-consuming discoverability search tool. However, despite being a last minute effort, a reasonable and highly functional quiz system was provided nonetheless.

PAM, described in the design chapter, was already a very useful and intuitive quiz system from both the instructor's and student's standpoint. It featured a method to easily create questions and answers, assign points to individual questions, perform automatic grading, set and enforce time limits for timed quizzes, and intuitively report the results of the quiz to both instructors and students. It also was very good from a security standpoint. It stored the answers to the quiz in the database in an encrypted form, encrypted the answers students gave on the quizzes when sending them over the network, and had several different and independent layers of code built in to ensure that students could only take a quiz once. It was therefore obvious from the start that PAM would not need any additional features built in, as would have been necessary for an application that was intended for doing web polling. I therefore realized that all that was necessary was to

modify it to be compatible with OfCourse's new plugin architecture, just as I had done with Promia. For more information on the specific details regarding the integration of PAM, I refer the reader to Appendix C.

This plugin, since it was new to the system, was again subjected to the IEEE 829-1998 test format, and was tested by the same five different testers mentioned previously. Those test results, recommendations, and bug fixes are found in Appendix E.

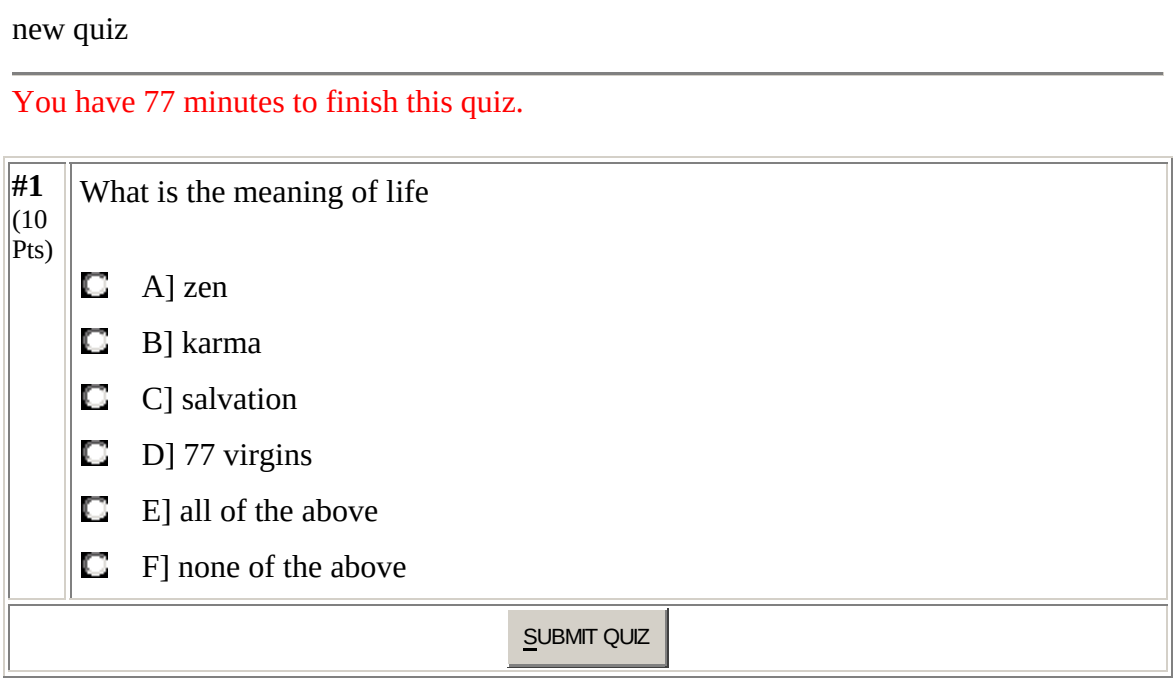

Figure 7 - Screenshot Of OfCourse's New Quiz System

4.3 - Results of the Discoverability Search Tool

I accomplished the discoverability search function by producing three distinct parts of the searching application. The first is the search API, which as previously mentioned, contains all of the necessary functions for producing, normalizing, weighting and sorting the discoverability vectors that are produced within the metadata spaces for the various searchable (or discoverable) components of the OfCourse system. The next component is the user interface, which is encapsulated within the search.cgi page. This script is essentially responsible for providing the user with the appropriate search

interface. For Basic search this is a simple text field and a search button, whereas for Advanced search the complete plethora of various inputs, selections, details and controls for custom-tailoring a fine-grained query is available. The third and final component of the search tool is the various search modules that are implemented to search the specific plugins themselves. Each searchable plugin, therefore, has its own search module, or code, that knows only of how to search the data contained within that specific plugin. In this way, when a user instigates a search, the query is passed into these plugin-specific search modules via the search API through the use of CGI environment variables. The specific search modules are then responsible for generating the results for their particular plugin, complete with details and hyperlinks as necessary, and presenting them to the user via HTML. Therefore, if OfCourse is extended in the future by adding an additional plugin, or revising and extending an existing plugin, the developer(s) only needs to write or revise a single plugin-specific search module. This provides a highly flexible application structure, as the user interfaces, discoverability vector computation functions, and plugin-specific search software are each separated and modularized into their own components.

The result of this is that now all of the data in OfCourse's myriad of plugins is now discoverable from a single cohesive and human-friendly search interface. No longer must the user methodically comb through each plugin, hoping by chance to stumble upon information of interest. Imagine how much more effective the OfCourse distance learning software has become with the addition of this new and exciting feature. The student can now quickly and easily locate relevant information from the discussion board, file exchange, events calendar, user directory, chat room, or static web based course content with no more effort than a single click of a mouse. The needed data, if found, is then spread out before the student in a comprehensive and well organized fashion, sorted in order of relevancy to his or her search. Needles to say, such an experience can vastly increase the pleasure and rate of learning when dealing with distance education. To further enhance the user's power of selection and choice, I developed OfCourse's search application to provide the user with two clearly distinct and powerful methods to conduct a query: Basic Search and Advanced Search.

4.3.1 - Basic Search

Basic search in OfCourse is very similar to other widely used Internet search engines such as Yahoo and Google. There is simply a text field to enter a query string and a search button, both of which should be excruciatingly familiar to most web users. The beauty of this search method is its simplicity, just enter the text, click the button and you get your results. Also, this text field and button was made to be a simple, two line HTML widget, able to be embedded anywhere throughout the system, and even outside the system, such as within the static HTML course web content. Basic search in its generalized form searches all areas of OfCourse, but through a simple URL CGI command-line option, it can be custom tailored to search only one part, or parts, of the system. Thus, the search box present in the discussion board performs searches on only the discussion board, the one within the events calendar only searches that plugin, and so on. Figure 4.5 shows the Basic Search user interface. While this method of search is indeed sufficient for most user requests, I also recognized that some users would want to take more control over their search process, and thereby their search results. This is why in addition to Basic Search I also added an Advanced Search feature.

Figure 8 - Basic Search HTML Widget

4.3.2 - Advanced Search

While taking slightly more time and effort on the users part, performing an Advanced search has several distinct advantages over Basic search. First, the user can select which specific areas of the system (plugins) they wish to search. Then, within each plugin, they can select various options as to which pieces of metadata that the query applies to. For example, if searching the discussion board, they can select whether to search the title only, content only, or both titles and content. Also, they can select if the query string is a list of keywords (default) or an exact phrase. This relates to the metadata

space tree described in earlier chapters. Each possible selection is searching within one (or more) particular objects on one (or more) levels deep within the object tree. This adds far more selectivity to the search, since a user can drill down as deep as they need to but only within selective areas of information contained throughout OfCourse.

Figure 9 shows the advanced search page. Notice how the search controls for each plugin are contained in their own separate unit on the page. This is once again an example of the modularization of the search system. Indeed, each plugin is responsible for delivering a module to provide its own customized advanced search controls if necessary. I developed each of the existing plugin's advanced search control code, as well as developed new code to search PAM. In doing so, I demonstrated how easy it would be for a future developer to implement their own search features and functionality for their own specific plugin. This, I believe, adds a high degree of flexibility and robustness to the structure of the code.

To make the static web course content searchable, software was needed to find and parse the web resources that the instructor has provided and put online. To accomplish this, I created a simple but highly functional and powerful web spider for parsing the HTML pages that make up the static course content. Using this spider, an instructor can crawl through each of the pages, starting with the root page on the main menu, and proceed in a breadth-first manner until he/she reaches the boundary, or frontier, of the course content (usually in the form of external links). The spider provides a full implementation of HTTP/1.1, as well as several regular expression language *patterns* that are used for eliciting the text of the web page, capturing hyperlinks embedded within the page (for further exploration, as necessary), and finding and extracting the page's metadata states, such as title and keywords. The careful construction of the patterns provided a quick and speedy way to implement a relatively complex parsing algorithm in just a few lines of code. The result is a tool that makes it very easy to make web pages external to OfCourse searchable from within OfCourse. A full range of advanced search controls, similar to those found at Yahoo and Google, were also provided to the user as a means of searching this selected set of course documents and web pages (see figure 9).

Search [ ] Search

☑ **Search Discussion Board**

Query String Is: A List Of Keywords

Search For Query String: In The Subject And In The Content

◉ All ○ Archived Only ○ Non-Archived Only ○ Selected Topics Only

Discussion Board Topics To Search:

- ☐ Lab 0
- ☐ Introductions
- ☐ Lab 1
- ☐ File Upload
- ☐ Startup Directions
- ☐ Surveys
- ☐ Lab 2
- ☐ Lab 3
- ☐ Turning assignments in
- ☐ Projects
- ☐ Lab 2b - Perl/CGI
- ☐ Audience analyses
- ☐ Lab 4
- ☐ lab 5
- ☐ Traveling
- ☐ lab 6
- ☐ Technologies
- ☐ Lab 7
- ☐ Lab 8
- ☐ Lab 9
- ☐ final and final grades
- ☐ Vijay
- ☐ Grades

Discussion Board Objects To Search: Threads and Posts

Results Should Be Dated: Any Date | Jan | 1 | 2000

From User: All

☑ **Search Events Calendar**

Search For Query String: In the Event Name And The Event Description

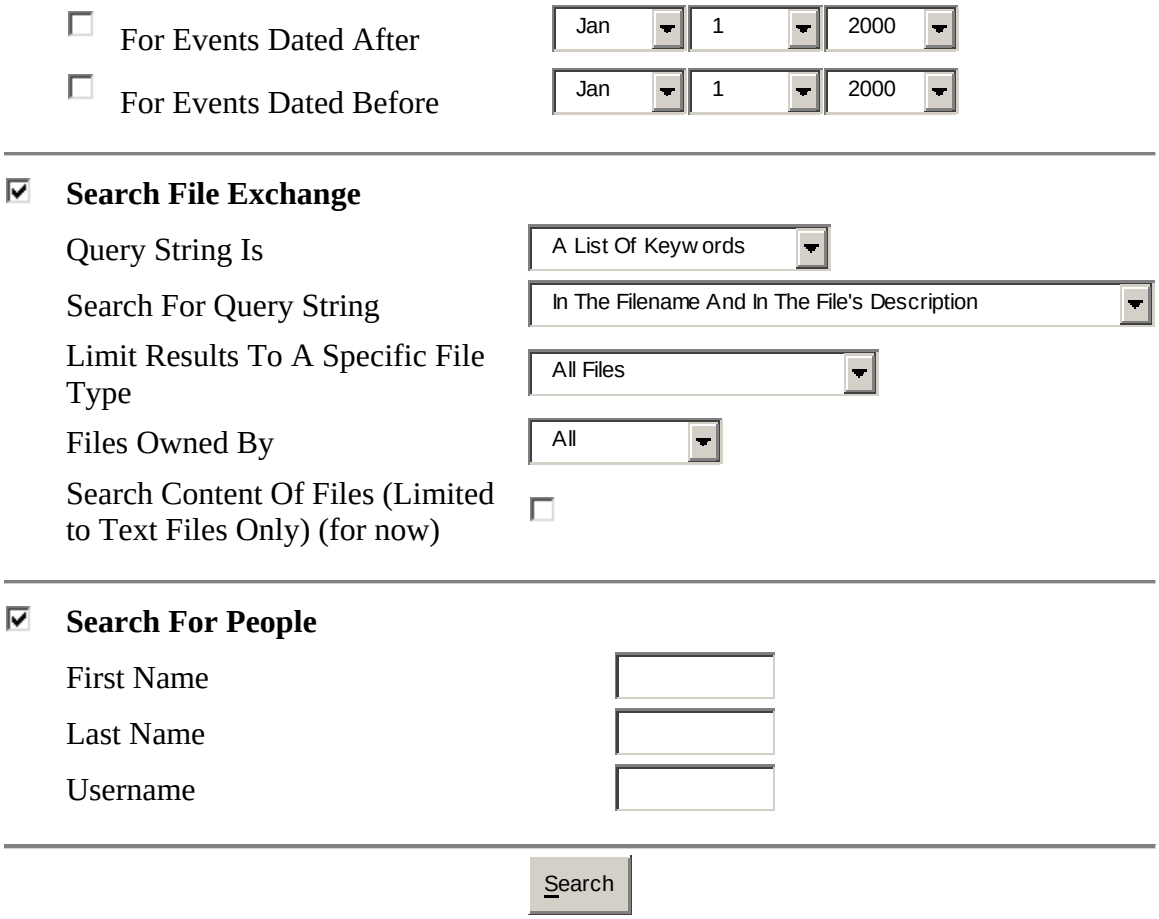

Figure 9 - Advanced Search

4.4 – Testing

For testing the new software, and the changes to the existing code base, I used the IEEE 829-1998 test format structure, which comes as highly recommended by software engineers for use by human based testers. I designed three separate IEEE 829-1998 tests for the testing of PAM, Promia and Discoverability Search and recruited human testers to perform these prescribed tests on the system. I then carefully analyzed their feedback and bug reports, and made adjustments and/or fixes to the software as necessary. A sampling of these reports can be found in appendixes D, E, and F. With the testing done and the bug fixes made, I felt that the new code and improvements to OfCourse could now be classified as “stable release” status for use in the field.

Professor Lemone and Seth Hunter were the first people to perform the tests, soon after the application was finished. They found several bugs, which were subsequently fixed. For example, within the file exchange, the directory path was improperly set, causing files that were uploaded to be placed in the wrong directory. In the quiz system, the auto-generated forms for creating quizzes were not being filled out completely. Also, quizzes that were previously deactivated could not be made active again. For the search plugin, there were some bugs found in the user interface, causing awkward display of search results. These were simple fixes and minor bugs from a developer's standpoint, but any one of them had the potential to make parts of the system behave strangely or fail to operate properly. Thus, from the first round of integration testing, several key bugs were identified and fixed long before the newly updated version of OfCourse went live. This, no doubt, has proven to be very beneficial.

More bugs were, however, found when the application was installed in a production environment. The vast majority of these bugs stemmed from the fact that there were stale modules of code that somehow found their way in to the final release. For example, an outdated version of both the chatroom and the file exchange were present in the final release. After looking into the issue on the SVN server, I discovered that an erroneous check-in had occurred sometime in the past, resulting in deprecated code being injected into the application structure. This situation was quickly remedied by replacing the suspect code with the correct versions, both in the production environment and on the SVN server.

There was one other bug discovered that was not related to this issue, though. The calendar plugin, which worked fine in the development environment, suddenly stopped working when it was placed into the production environment. After some poking around, I found out that a blank line present in the header of a PHP script file was causing the problem. This was not really a bug per se; it was simply an incompatibility between the PHP interpreter on the development machine and the PHP interpreter on the production machine. Nevertheless, this was a significant issue, as it caused the calendar plugin to fail completely. After making this change, I once again tested it in the development environment to see if it would still work there. It did. Thus, I found out through testing that one brand of PHP interpreter ignored blank lines while another brand interpreted

them as code. I made a note of this in the readme file of the install package, to alert future users of the software to this anomaly.

Finally, after the application was in production, several more testers were recruited from Professor Lemone's CS4533 Compilers Class. They performed the same IEEE 824-1997 tests mentioned previously that were made up for the various plugins. Tables detailing their feedback are presented in Appendices D, E, and F. Their recommendations for improvements to the system are presented in the Conclusions chapter, which follows this one.

4.5 - Conclusion

The goals of this project were to 1) modularize OfCourse into a plugin-API style architecture 2) find major security holes and fix them 3) redesign and/or replace existing plugins to make them more useful to the user and 4) to implement a functional and cohesive system wide search tool. This chapter has explained the results of achieving those goals. I replaced the awkward functions.cgi with the new user, database, and admin APIs to make the software structure second to none. Throughout the chapter, I explained how making those critical changes resulted in vast security fixes and improvements. I found, evaluated, and re-designed PAM and Promia to be used as plugins to OfCourse. Finally, I explained the results of the highly advanced and powerful discoverability search tool and the new web spider. All of these results, I believe, have culminated into a vast and remarkable improvement to OfCourse in many, many ways.

5. Conclusion and Recommendations

The mission of this project was enhancing the distance education application called OfCourse. Specifically, these goals were the following: Modularize the software structure into a API-plugin architecture, find major security holes and fix them, replace and/or redesign existing plugins to make them more functional and user-friendly, and, most importantly, develop and implement a system wide discoverability search tool. While all of those goals were achieved completely during the six-week time frame allotted to this project, there were also several notable suggestions for future improvements from a range of sources including the individuals who tested this software, Professor Lemone, and myself. A description of their recommendations as well as a summary of the enhancements made during the project is the subject of this chapter.

5.1 - Summary

At the beginning of this project, OfCourse was a relatively functional software application on the surface. However, underneath the hood it was perhaps a few steps up from a total disaster. The application structure was highly disorganized, with multiple co-dependencies and obsolete libraries being used throughout. It required the entire code base to be reinstalled for each new course that an instructor wanted to teach. There were several glaring security holes that could enable crackers to easily gain administrator privileges, such as the sending of passwords in clear text. Some of the tools, such as the quiz system, were totally inoperable due to bugs, while others, such as the file upload tool, were very awkward for the average computer user to figure out how to use. However, perhaps the most glaring omission to the whole application was the complete absence of a search facility. I focused on each one of these goals in turn.

First, I concentrated on reorganization of the software structure. I developed several top-level APIs, including the database API, user API, admin API, login API and search API, then restructured the existing modules of code into plugins for the new application program interface. While doing this, I naturally came across the numerous security holes present in the system, then designed and implemented fixes for them. After

this was accomplished, I then focused on redesigning and/or replacing certain plugins that were holding OfCourse back in terms of ease of use and/or functionality. Specifically, these included the file exchange, chat room, and quiz system. After locating suitable candidates from the open source community, I modified these third-party solutions to plug in to OfCourse's new APIs. This made vast improvements to these areas of the system. Finally, yet most importantly, I designed and implemented the system-wide search feature that OfCourse now contains. To do this, I looked to the concept of Watson and Wiley's Discoverability Search and built an object-oriented semantic structure called a metadata space around the data in OfCourse that was to be made searchable, then implemented appropriate algorithms to search the resulting metadata space as necessary. The result is a powerful and flexable search tool that allows users to find relavent information all throughout OfCourse, from both a basic search and advanced search interface.

During this project, I used all the phases of the software development cycle to enhance OfCourse into a well organized, easy to use, secure, and searchable distance learning application. Each of these enhancements made an already good piece of software even better. Indeed, at the conclusion of this project, OfCourse is currently installed and running in a production environment, serving the needs of an online classroom.

5.2 – Recommendations

- Create a backup feature so that the database can be backed up at regular intervals
- Make user interface improvements to the discussion board. Give it the look and feel of a modern forum
- Improve administrative controls, giving the administrator more fine grained control over the user interface display
- Add a video archive to the chat room, so that students can review lectures from the past.
- Incorporate a help feature or link for each page within OfCourse.

- When logging in, display the user's last login date and time like a UNIX system does. This can alert users to unauthorized access.
- Link the quiz system directly to the gradebook, so that when quizzes are taken the student's grade is recorded automatically.
- Have the system email the administrator when more than three unsuccessful login attempts happen from the same IP.
- Allow administrators to maintain a list of banned IPs
- Have a turnin feature that is separate from the file exchange that is only used to submit assignments
- Add to the chatroom the ability to create private rooms such that users can have private chats with one another.
- Fix the smilies in the discussion board
- Allow users to upload their picture to the user directory, so that other users can put a face (or an avatar) to a name. Do the same in discussion board posts.
- Make certain error messages are as non-technical as possible
- Incorporate a two-way radio feature into the chatroom.
- Add a whiteboard facility to the chatroom

6. Bibliography

Williams, Marcia L. (1999) Distance Learning: The Essential Guide. Sage Publications: Snata Clara, California. 0761914412

Describes how to integrate familiar applications into the online classroom making users feel comfortable; also includes trends in distance learning and the online classroom.

Primary Research Group, Inc. (1997) The Survey of Distance Learning Programs in Higher Education. Primary Research Publishers: New York, New York. 1574400088

A statistical review of distance learning students, teachers and methods, containing many charts of raw statistical data as well as some descriptions of the data.

Pallof, Rena M. & Pratt, Keith (1999) Building Learning Communities in Cyberspace Jossy-Bass Sansome Street Press, San Francisco, California

A guide to conducting experiments with distance learning, transitioning from a traditional classroom, protecting privacy and ethics in the online classroom, and an evaluation of the effectiveness of online teaching techniques.

Berge, Zane L. & Collins, Mauri P. (1995) Computer Mediated Communication and the Online Classroom. Hampton Press: Cresskill, New Jersey. 188130308X

A book which gives ideas about various searching strategies for the online classroom, how to design an online classroom and trends in distance education for the 21st century.

Petrides, Lisa A (2000) Case Studies On Information Technology in Higher Education Idea Group Publishing: Hershey, Pennsylvania

Gives many case studies on course management systems and techniques. Performs a cost/benefit analysis of various online learning techniques and provides an example of building a relational database for an online education system.

Starr, Roxanne H. (2000) The Virtual Classroom: *Learning Without Limits* Ablex Publishing: Norwood, New Jersey.

Explains many aspects of human/computer interaction with regards to distance learning applications. Also gives tips and attempts to bridge the gap in transitioning from traditional to online education.

Meyer, Katrina A. (2002) Quality in Distance Education: Focus on Online Learning. The George Washington University Press: Washington, DC. 0787963496

Researches the impact of distance learning technology on students and faculty and their attitudes. Describes means of measuring quality, eliminating misunderstandings and charts the rapid growth of distance education.

Ko, Susan & Rossen, Steve. (2001) Teaching Online: A Practical Guide. Houghton Mifflin: Boston, Massachusetts. 0618000429

Describes methods of constructing online classrooms and course management systems; also offers information on how to convert existing course material to an effective online format.

Khan, Badrul H. (2003) Web Based Instruction Educational Technology Publications: Englewood Cliffs, New Jersey

Describes how to best design web based instruction courses, user interface design, supporting collaborative learning and methods on adapting traditional education material to the world wide web.

McClure, Polly A. (2003) Organizing and Managing Information Resources On Your Campus John Wiley & Sons: San Francisco, California

On effective management techniques of online course management systems, dissemination of information to students and online help and training for these systems. Provides detailed guide on how to select an appropriate course management system for university level education.

Blackboard (2005) Retrieved August 28, 2006 from http://www.blackboard.com/

Home page for the popular commercial course management system Blackboard; gives information about and demonstrations of the software.

Peraya, Daniel (2006) "Distance Education and the WWW" Retrieved August 28, 2006 from http://tecfa.unige.ch/edu-comp/edu-ws94/contrib/peraya.fm.html

Gives an overview of the current state of distance learning and how to use the WWW to most effectively teach online. Discusses both the theory and practice of using the web to design online learning courses.

Sherry, L. (1996). Issues in Distance Learning. *International Journal of Educational Telecommunications, 1* (4), 337-365. Retrieved August 28, 2006 from http://carbon.cudenver.edu/~lsherry/pubs/issues.html

Journal article describing several issues related to distance learning, how to select the proper technology and how to measure the quality of the online educational experience from both the teacher and student's point of view.

Course Management Systems: *A Guide to Researching Product Features of Over 40 Different Course Management Systems.* (2006). Retrieved August 28, 2006 from http://www.edutools.info/static.jsp?pj=8&page=HOME

Web site that offers a tool to do side by side comparisons of the features of over 40 different commercially available course management systems.

Moodle: *A Free, Open Source Course Management System For Online Learning* (2006). Retrieved August 28, 2006 from http://moodle.org

Home page of the popular open source distance learning application Moodle. Provides documentation of the software as well as the source code itself.

Angel Distance Learning Software (2006) Retrieved August 28, 2006 from http://www.angellearning.com/

Home page of the course management system ANGEL. Provides software documentation, online help, and an extensive FAQ.

WebCT eEducation Software (2006) Retrieved August 28, 2006 from http://www.webct.com/

Home page of the commercially available proprietary course management system WebCT, a direct competitor to the Blackboard system annotated above.

Manhattan Open Source Course Management System (2006) Retrieved August 28, 2006 from http://manhattan.sourceforge.net/

Another open source distance education application. Provides features similar to the Moodle project.

Sakai Project: *Collaboration and Online Learning Environment* (2006) Retrieved August 28, 2006 from http://www.sakaiproject.org/

Homepage of the Sakai project, which is a freeware version of a popular web based distance education and course management product.

ILIAS: Open Source Software for Distance Education (2006) Retrieved August 28, 2006 from http://www.ilias.de/ios/index-e.html

Home page to ILIAS, an open source online learning application. Provides documentation, user reviews, explanations of the source code, and articles written by the developers of the package.

Gilmore, Jason W. (2004). Beginning PHP 5 and MySQL. Sage Publications: Santa Clara, California 1893115518

Book providing a beginner's guide to programming in PHP. Gives examples and methods of using PHP and MySQL to create web based relational database applications.

Lerdorf, Rasmus & Tatroe, Kevin (2006). Programming PHP. O'Reilly Media: New York, New York 0596006810

O'Reilly book for professional PHP programmers. Demonstrates coding examples, techniques and advice to the computer scientist that wants to use PHP to create dynamic web based content.

Watson, Thomas and Wiley, David (2001). Structured Metadata Spaces. *Journal Of Internet Cataloging*

Scientific journal article by the original developers of discoverability search. Explains discoverability vectors, metadata spaces, search strategies, object representations and other concepts relevant to the construction of discoverability search tools.

Watson, Thomas (2005). Lessons on Effectively Optimizing Content for Internal and External Discoverability. *Special Libraries Association, Fairfield Counties Chapter*

A live presentation by Thomas Watson, co-inventor of discoverability search. Presents informative lecture and discussion about lessons learned while implementing discoverability search in a wide range of real-life sceneries, such as libraries, cooperate data warehouses, government databases, and the World Wide Web.

Appendix A – Creating Digital Media For The Electronic Classroom

One of the issues that must be addressed is which types of media the instructor wishes to use to educate her/his students, and how to best present these in an electronic form for use over a network. For the purposes of most learning environments, these typically include text (i.e.: books, articles, notes), audio (e.g.: lectures) or video, although combinations are also possible. Methods of and issues with the conversion and presentation of these three vital methods of human communication in electronic form will now be explored.

The use of text based media in the digital classroom is practically a given. The syllabus, the schedule, notes, drawings, books and articles just to name a few will almost certainly fall under this category. The best methods to convert the existing material to a form that is usable on a computer vary considerably, but they should all have a few things in common. First and foremost, they should aim to be converted into a media type (file format) that is compatible with as many different types of computer systems as possible. This is true not only with text media, but with all the other types of media as well. While it might be tempting to use *.doc files for class notes and other text materials, for example, one should always keep in mind that the student's computer may not have that particular word processor installed, or be using an alternative operating system, such as Linux or Mac OS X. A choice such as rich text format (*.rtf) or HTML would be mush more flexible, and provide most of the layout features offered by proprietary software. Another issue to be considered here is how time consuming it is to convert information on paper, say, into the desired file format. While there do exist several programs that take a image file of text on paper (produced by a scanner hooked up to a PC, say) and extract the text to a textual based file format, they tend to vary wildly in quality and accuracy. It seems that few things can be more frustrating to a student trying to learn than to have to decipher a document that has not been decoded properly by the software, and contains numerous errors in the text as well as in the layout. For this reason alone, not to mention the many others, all textual material converted to electronic form should be carefully proofread by the instructor(s) before being made available to the students.

In a similar nature to textual components, there are many features and formats available for audio and video data as well, but a careful balance must be achieved between quality, compatibility and file size. With audio/visual files for example, it seems that for every different proprietary digital media player available, there is at least one (and often many more) proprietary file formats that come with them. An entire volume could easily be written comparing and contrasting the various audio and video data file formats and structures, so only a very simplified general overview of them will be presented here. Some are better at reproducing quality than others, usually at the expense of file storage size, and are usually the product of lossless compression algorithms (programs that don't throw away any audio or video information during the compression process). Others are very good at compressing large, complex audio/video streams into small compact files for a less disk storage footprint and reduced network bandwidth requirements. These are usually produced by lossey compression algorithms, which in contrast to lossless compression, do actually throw away some of the input information, albeit usually an unperceivable amount, during their compression process. In addition to these considerations, there is the issue of whether or not any copyright protection is needed for the digital audio or video. While some file formats offer very high levels of encryption, authorization, and copy protection, others offer none of this whatsoever. One last important consideration that specifically applies to the subject of audio and video files is the method in which they will be disseminated. For example, the choice of media and how much of it to use can vastly depend upon whether the distance learning community will consist entirely of students and instructor(s) hooked up to a high speed network, for example, or a more heterogeneous community where some users can connect only at 56kbps (or slower) speeds.

Appendix B – Ethical Issues In Distance Learning

While in the past various methods of long distance communication have been used in the implementation of distance education, the Internet appears to be the way to proceed in the 21st century and beyond. However, there are several important ethical issues that remain to be considered when using such a powerful, highly interconnected worldwide network.

One of the most important of these issues to be considered is the issue of how the students and teachers will access the Internet, in particular with regards to speed and bandwidth. If everyone has a fast broadband connection, obviously more vast and expansive Internet technology can be deployed. This includes streaming audio and video, Voice over IP, and video conferencing, just to name a few. However, an important question arises when and if some students have the high-speed access while others do not. Should the higher bandwidth features be made available for those with the capacity to use them, even if this means that students with slower connections will be missing out? Clearly this can put a certain population of the students at an unfair disadvantage, if they live in an area where high-speed access is not available or are not able to afford the broadband Internet service. This issue is known to experts in the field of distance education as the Digital Divide.

The Digital Divide is the amount of social and economic separation between those who have access to computers and the Internet, and those that do not. As with the introduction of any new product, the computer has proliferated into modern society by first being the domain of the very rich, then gradually being adopted by the mainstream. However, there are still vast segments of the population who either have no access to, cannot afford, or are otherwise prevented from using, computer technology. These groups of people find themselves on one side of the Digital Divide, while the one's who have afforded and embraced the computer find themselves on the other. As distance learning today carries the prerequisite of computer access and literacy, people on the wrong side of the divide see fewer educational opportunities. On the other hand, those who are familiar with computers and are able to afford them, are finding more and more. Thus, a

simultaneous upward and downward sociopolitical and socioeconomic shift has been created by the emergence of this digital divide.

The Digital Divide exists between several key segments of the population. First, there is a major separation between the rich and the poor. This is the biggest divide in America, with people earning over 75,000 dollars a year being four times more likely to have a computer with high speed Internet access than people making 20,000 or less (U.S. Census Bureau, 2001). Also, there is much separation between the races in terms of computer access. A huge chasm has developed in the statistics, with Whites and Asians being twice as likely to have a computer in their homes than Blacks and Hispanics. Also, there is separation between the old and the young, with those who "grew up" with computers being more comfortable with using them. However, these domestic digital divides pale in comparison to the worldwide digital divide, which exists between developed and non-developed countries. "Most people in the world have never made a telephone call, let alone used a computer" (Sara Baase, 2002). While there are many viewpoints about the relevance of this data, as well as what to do about it, most experts do however agree on one thing: Those with little or no access to computers and the Internet have little or no access to distance education curricula. While this project obviously cannot address this large-scale worldwide problem, we must keep it in mind when considering issues such as file sizes, bandwidth requirements, and deployment of brand new cutting edge technologies within OfCourse.

Appendix C – Details Regarding the Integration of PAM and Promia

Promia

After the new plugin was integrated, I disabled much of the file and directory permission management features, and enabled only the basic user interface skin, so that Professor Lemone could test the application in its most basic form and determine which features of the file and directory permissions and user interface she wished to have turned on. Much to my surprise, she liked the plugin just fine the way it was, sighting its simplicity and easy learning curve as major advantage to its future users. I therefore decided to leave the more advanced features turned off, but they remain available to be activated through a text based config file if the need (or desire) ever arises to enable them in the future.

The last modification that needed to be made to Promia (now and hereafter referred to as OfCourse's file sharing plugin) was to add a way to associate user-specified keywords and or descriptions to the uploaded files. This would allow the files to be searchable by their keyword and description, instead of just their name that is sometimes not descriptive enough to describe the contents of an entire file. For example, the name of this file you are reading now is named OfCourseMQPreport.doc, however as I think you would agree, that name alone does a very poor job of describing the contents of the file. To accomplish this, I altered the "files" table in the database to include a textual column called “keywords.” I then modified the user interface to the file upload page to include a web-form text field for users to enter keywords and/or a description of their file prior to uploading it. A screenshot of the resulting file upload page is shown in Figure 4.3.

PAM

PAM was originally designed to work with a separate web based user management system, which made the modifications to accept OfCourse's user API calls a snap. The database connectivity required slightly more work, as PAM was originally

designed to use a Microsoft Access database via ODBC drivers, instead of the MySQL database that OfCourse uses. For this reason, most of the database queries had to be changed, as it turns out that Access uses a slightly different (and of course incompatible) SQL dialect than MySQL does. This was not as hard as it sounds, though, because all of PAM's database queries were properly separated from the user interface code. Indeed, they were all self contained in one single perl module, so that once this module was properly modified to be compatible with MySQL, the rest of the application did not know the difference. The result is a fully modified version of PAM that I easily plugged in to OfCourse's APIs.

Appendix D – A Test Of Promia

Test Plan (IEEE 829-1998 Format)

Test Plan Identifier

OC rev.126 12-06-06 #1 (file exchange).

Introduction

This procedure is to test the primary functionality of the file upload facility of OfCourse. The ability to properly upload text and binary files to the user's filespace, the ability to share these files with other users, as well as the ability to download the files without errors or corruption will be tested by this procedure.

Test Items

1. File Upload page will be tested for the following criteria of functionality
   - proper loading of the page
   - ability to name the file, as well as supply a list of keywords/description
   - ability to browse the local filesystem for a file to upload.
   - after upload, the ability of the user to return to the File List Page

2. File List page will be tested for the following criteria of functionality
   - proper loading of the page
   - properly displays links to all files that have been uploaded by the user
   - properly displays links to all files that have been shared with the user
   - gives users the option of sharing the file with other users

3. File Share page
   - proper loading of the page
   - proper listing of all users of the system in the dropdown choice menu
   - omission of the currently logged in user from the choice menu
   - ability to add shares to the file
   - ability to remove shares from the file

4. File Download
   - proper downloading of text based files without corruption
   - proper downloading of binary based files without corruption

Approach

Item #1: Navigation To File Upload Tool

1. user browses to OfCourse and selects awt06 course without logging in
2. user browses to the file upload tool.
3. after being told that a login is required, user logs in as "klemone"
4. user again browses to the file upload tool.

Item #2: File Upload Page

5. user clicks on "Upload A File" button at the top of the page
6. user names the file in the filename box "texttest1"
7. user enters a description of "testing text files" in file description field.
8. user clicks on "Browse" button
9. After the file dialog appears, user selects a text-based file from her hard drive
10. user clicks on "Upload File" button.
11. user clicks on "OK"

Item #3: File List Page

12. user verifies the presence of the new file in her filespace.
13. user clicks on "Add Shares" link next to the new file.

Item #4: File Share Page

14. user finds and selects "jwf2" from the dropdown box
15. user clicks "Add" button.
16. user sees "jwf2" added to the list of users this file is being shared with
17. user finds and selects "hunnr" from the dropdown box
18. user clicks "Add" button.
19. user sees "jwf2" added to the list of users this file is being shared with
20. user returns to the File List Page, by clicking on the "file exchange" button.

Item #5: File List Page (shares)

21. user logs off "klemone" and logs in as "jwf2" password "jwf2"
22. user returns to file exchange.
23. user sees "texttest1" file listed under the "files shared with you" category.
24. user clicks on the link to the file.
25. the file dialog appears, user enters a name for the file on local filesystem.
26. the user clicks "Save" in the file dialog box.

Item #6: File Integrity

27. user opens the file on the local filesystem, and verifies that it is not corrupted

Item #7: Binary File Upload

28. user logs off "jwf2" and logs in as "klemone"
29 user repeats steps 5 through 27, substituting a binary file for the text file.

Item Pass/Fail

Please circle either Pass or Fail for each item in the list above. If you circle Fail, please give a brief description of why the test failed, including any error messages that you may have received from OfCourse or the system itself.

Item #1 (Pass) (Fail)
Notes: some changes to wording needed
done them check that
Like this

Item #2 (Pass) (Fail)
Notes: ditto

Item #3 (Pass) (Fail)
Notes: sharing worked (maybe)

Item #4 (Pass) (Fail)
Notes:

Item #5 (Pass) (Fail)
Notes:

Item #6 (Pass) (Fail)
Notes:

Item #7 (Pass) (Fail)
Notes:

Test #1 of Filesharing Plugin		
ITEM #	**PASS/FAIL**	**COMMENTS**
1	Pass	
2	Pass	
3	Pass	
4	Pass	
5	Pass	
6	Pass	
7	Pass	
General Comments	N/A	Seems Promising
Recommendations	N/A	It is slightly annoying that after you upload a file it has that separate confirmation page, rather than just taking you back to your fileshare and mentioning that you successfully uploaded a file. It isn't a big deal though.
Compare to myWPI	N/A	The big difference is that when saving a file from myWPI it will fill in the proper name and file extension for the suggestion as what to save the file as, rather then manually having to change "blah.cgi" to a proper name.

Test #2 of Filesharing Plugin		
ITEM #	**PASS/FAIL**	**COMMENTS**
1	Pass	
2	Pass	
3	Pass	
4	Pass	
5	Pass	
6	Pass	
7	Pass	
General Comments	N/A	naming of files is not strict. /'s ;' etc are allowed.
Recommendations	N/A	when trying to download / save as an uploaded file it gets renamed to "postal.cgi" uploading a file with name ; says it is successful but does not show the file.
Compare to myWPI	N/A	

Appendix E – A Test Of PAM

Seth

Test Plan (IEEE 829-1998 Format)

Test Plan Identifier

OC rev.126 12-06-06 #3 (Quiz System).

Introduction

This test will verify proper operation of the OfCourse quiz system. Since this is third party software in Production/Stable release status, I believe that much testing has already been done on this piece of the system. However, there were some extensive modifications that needed to be performed to adapt the quiz package to the plugin style architecture of OfCourse. For this reason, I have provided the following test procedure to verify the correctness and functionality of the software.

Test Items

1. The ability to create quizzes
2. The ability to add questions to quizzes
3. The ability to edit quiz questions.
4. The ability to activate and deactivate quizzes
5. The ability to take quizzes
6. The ability to obtain the results of quizzes

Approach

Item #1: Quiz Creation — URL ?

1. User logs in to awt06 course as "klemone" - password?
2. User clicks on ~~Class~~ Tools button.
3. User clicks on "Create/Modify Quizzes" button
4. User enters "Quiz 2" into the quiz name field.
5. User enters a time limit of 60 min in the time limit field
6. User clicks "Create Quiz" button.
7. User verifies that "Quiz 2" now appears in the list of quizzes
8. User verifies that "Quiz 2" is de-activated by default.

Item #2: Adding Questions to Quiz

9. User clicks on the "Edit" button next to "Quiz 2"
10. User enters text "question one test" in the text field. - I see 8 text fields
11. User clicks "CREATE QUESTION" button.
12. User verifies the following error messages from the form:
    - Invalid points entry
    - Must Supply at least two answers to the question
    - Must supply a correct answer to the question
13. User enters a non-negative integer for the points field
14. User enters "ans1 correct" for answer "A"

15. User enters "ans2 wrong" for answer "B"
16. User clicks the radio button next to answer "A" selecting it as correct
17. User clicks the "CREATE QUESTION" button again
18. User verifies the question was created and appears in the list below

Item #3: Editing Quiz Questions

19. User clicks the "edit" link next to the question just created
20. A pre-filled-out form is displayed with the information previously supplied
21. User adds "ans3" to the answer for question "C"
22. User modifies the question text to read "question one test edit"
23. User clicks the radio button next to answer "A" selecting it as correct
24. User clicks the ~~submit~~ button on the form. — create question
25. User verifies the modifications just performed have taken effect.

Item #4: Activating and De-activeating quizzes.

26. User returns to the "Administer Quizzes" page — How? yes, how?
27. User confirms the "Quiz 2" is still deactivated. ?
28. User clicks the "Activate" button next to "Quiz 2"
29. User verifies that "Quiz 2" status is now "Active (1)" — OK, but can the UI be more friendly?

Item #5: Taking Quizzes Course

30. User clicks on "~~Class~~ Tools" button on the toolbar
31. User clicks on "Take Quizzes" link.
32. User clicks on "Quiz 2" from the list — The UI should warn the user that the timer starts as soon as they click the link.
33. User verifies that "Quiz 2" is properly displayed
34. User answers "A" for question one.
35. User clicks on "Submit Quiz" button at the bottom of the page.

Item #6: Obtaining Quiz Results

36. User verifies her score 1/1 100% for questions
37. User verifies that she recieved all of the points possible for the quiz. redundant?
38. User clicks on "~~Class~~ Tools" button on the toolbar.
39. User clicks "View Quiz Results" from the admin menu.
40. User sees one turnin for "Quiz 2"
41. User clicks on "View Results" button next to "Quiz 2"
42. User verifies her score was properly recorded next to her name.

Item Pass/Fail

Please circle either Pass or Fail for each item in the list above. If you circle Fail, please give a brief description of why the test failed, including any error messages that you may have recieved from OfCourse or the system itself.

Item #1 ~~(Pass)~~ (Fail)
Notes: _____
_____

Item #2 (Pass) (Fail)
Notes: Instructions said a positive number, I entered 12345678909876543211. It failed. I entered 25, it worked but I had to re-select correct answer.

Item #3 (Pass) (Fail)
Notes:

Item #4 (Pass) (Fail)
Notes:

Item #5 (Pass) (Fail)
Notes: maybe some user create 2 Q's & to 1 correctly, 1 WRONG

Item #6 (Pass) (Fail)
Notes:

Environmental Needs

(none)

Staffing And Training Needs

(none)

Test #1 of Quiz Plugin		
ITEM #	**PASS/FAIL**	**COMMENTS**
1	Pass	
2	Pass	
3	Pass	The edit form did not show that A was still the correct answer (the radio button was not selected).
4	Pass	I assume that the "Create/Administer Quizes" link you refer to should really be "Create/Modify Quizes" as there is no "Create/Administer Quizes".
5	Pass	
6	Pass	I assume that the "Class Tools" button you refer to should really be "Course Tools" as there is no "Class Tools" button.
General Comments	N/A	
Recommendations	N/A	
Compare to myWPI	N/A	

Test #2 of Quiz Plugin		
ITEM #	**PASS/FAIL**	**COMMENTS**
1	Pass	
2	Pass	"There were some errors while prosessing your POST" Not all users are aware of what POST means, perhaps make the error msg less technical?
3	Pass	
4	Pass	
5	Pass	
6	Pass	-Quiz's with duplicate names show separately in the create/edit view but only one shows up in the 'take quiz' view. -After hitting "OK" at the end of the quiz, the site loads in a frame, inside another frame. So there are two toolbars across the top showing "main page" "login/logout" etc...
General Comments	N/A	
Recommendations	N/A	

Compare to myWPI	N/A	

Appendix F – A Test Of Search

Seth

Test Plan (IEEE 829-1998 Format)

Test Plan Identifier

OC rev.126 12-06-06 #2 (Search).

Introduction

This test will verify the core functionality of the search facility of OfCourse. The search facility is capable of searching the discussion board, file exchange, user directory, calendar system, chat room, and course web content.

Test Items

1. The ability to search the discussion board will be tested
   - Ability to search each topic
   - Ability to search each thread
   - Ability to find individual posts

2. The ability to search the file exchange will be tested
   - Ability to find files based on keyword
   - Ability to find files based on filename
   - Ability to discover only the files that are shared with you

3. The ability to search the user directory will be tested
   - Ability to quickly and effeciently locate people in the directory

4. The ability to search the calendar system will be tested
   - Ablity to find events within the event calendar

5. The ability to search the course web content will be tested
   - Ability to find web pages based on keyword or subject search

Approach Or: note URL, select class! http://klaatu.cs.com/jwf/ofcourse/ocourse.cgi

Item #1: Searching the Discussion Board

1. user logs into OfCourse ~~awf06~~ class4 as "jwf2" password "jwf2"
2. user clicks on "~~class~~ course tools" button, then "Search OfCourse" link.
3. in the search field, user types in "seth" and clicks "Search" button.
4. user sees 5 results from the discussion board.
5. user verifies that the first result contains "seth" in both title and content.
6. user verifies that the first result has the highest discoverability rating.
7. user verifies the keyword "seth" is highlighted where it appears in the content.

adv search

Item #2: Searching the Event Calendar

8. user types in "event" to the search box.

9. user clicks "Search"
10. user sees one result, the "test event" event in the calendar
11. user verifies discoverability is 100 percent.

Item #3: Searching the File Exchange

12. user enters "test file" into the search box
13. user clicks "Search"
14. user scrolls down to results from the file exchange
15. user verifies one hit "test - this is a test file" from file exchange
16. user verifies that discoverability is 86 percent

Item #4: Searching the User Directory

17. user enters "klemone" into the search box
18. user clicks "Search"
19. user scrolls down to the results from the user directory
20. user verifies that the only hit for the user directory is "Karen Lemone"
21. user verifies that the contact information supplied is correct
22. user verifies that the mailto link provided is a valid email — How?
23. user clicks on the mailto link and verifies that it works

Item #5: Searching the Static Course Content

24. user enters "mqps" into the search box
25. user clicks "Search"
26. user scrolls down to the results from the user directory
27. user verifies one result
28. user clicks on the link provided.
29. user verifies that the link took her to the proper page
30. user verifies keyword "mqps" appeared in that web page — No - "mqp's"

Item Pass/Fail

Please circle either Pass or Fail for each item in the list above. If you circle Fail, please give a brief description of why the test failed, including any error messages that you may have recieved from OfCourse or the system itself.

Item #1 (Pass) (Fail)
Notes: "Course Tools", Give URL to user, course id incorrect

Item #2 (Pass) (Fail)
Notes:

Item #3 (Pass) (Fail)
Notes:

Item #4 (Pass) (Fail)
Notes:
22 - tell user how to do this

Item #5 (Pass) (Fail)
Notes:
30 - "mgps" does not, "mgp's" does

Item #6 (Pass) (Fail)
Notes:

Item #7 (Pass) (Fail)
Notes:

Environmental Needs

Staffing And Training Needs

Test #1 of Search Plugin		
ITEM #	**PASS/FAIL**	**COMMENTS**
1	Pass	
2	Pass	
3	Fail	There were no results from the File Exchange. The search found multiple discussion board postings and the test event from the previous test (with discoverability 70%).
4	Pass	
5	Pass	
General Comments	N/A	Seems Promising
Recommendations	N/A	Looks a bit rough, but nothing seems fundamentally wrong.
Compare to myWPI	N/A	From the student-user point of view, myWPI is much nicer. However after getting used to OfCourse it seems to be as easy to use as myWPI

Test #2 of Search Plugin		
ITEM #	**PASS/FAIL**	**COMMENTS**
1	Pass	
2	Pass	
3	Pass	
4	Pass	
5	Pass	
General Comments	N/A	It works okay, it takes some getting used to though.
Recommendations	N/A	An icon or text next to each hwk assignment letting users know if it is 'up to date' or not.
Compare to myWPI	N/A	Seems to have fewer features and a smaller simpler feel to it -It is not integrated w/ banner -It is not integrated w/ STUDENT domain -The submission system used in both myWPI and ofCourse is frustrating to work with. (the 'turnin' webapp was very nice though)