**YeiSol Woo**

**CS 460 Senior Capstone Project**

**Prof. Tanik**

**Exam II**

**Questions 1-4**

1. Make a 1-page overview table which lists

a. Your project roles/titles

b. Tools used

c. Tasks assigned

d. Tasks completed

|  |  |
| --- | --- |
| Project roles/titles | Web developer, Business logic developer |
| Tools used | Eclipse, Cmap, MS Visio, MS Project, MS PowerPoint, MS Word |
| Tasks assigned | 1. Usecase specification for user authentication
2. Update group Cmap
3. Create a user login page
4. Display a bingo table
5. Add tab design to display bingo sheet
6. Create bingo JSP page
7. Create the general website template
 |
| Tasks completed | I completed all tasks assigned to me. Since I am managing the whole template of the website, I will update some stuff on the website continuously. |

2. Make a table of tasks completed and describe the following for each task in table format (see Gantt Chart for details):

a. Task name

b. Task description

c. Task start and end dates

|  |  |
| --- | --- |
| Task name | Usecase for user authentication |
| Task description | Create a Usecase model and Usecase specification for user authentication by using RUP template |
| Task start and end dates | 1/25 – 2/5 |

|  |  |
| --- | --- |
| Task name | Update group Cmap |
| Task description | Update the overall structure of our group Cmap, add all the links that we have so far, and color-coding the objects |
| Task start and end dates | 1/25 – 2/5 |

|  |  |
| --- | --- |
| Task name | User login page |
| Task description | Create a simple webpage to verify the user's credentialsIf a user provides right information, LDAP bind is successful. Otherwise, the LDAP bind is unsuccessful then the credentials are considered invalid and no session is created for the user.  |
| Task start and end dates | 2/9 – 2/14 |

|  |  |
| --- | --- |
| Task name | Display a bingo table |
| Task description | Create a bingo table in HTML to display “B.S. CS Bingo Sheet” (Without Tab Design) |
| Task start and end dates | 2/22 – 3/13 |

|  |  |
| --- | --- |
| Task name | Add tab design to the bingo sheet |
| Task description | Updated the previous bingo table by adding tab design to display “B.S. CS Bingo Sheet” efficiently |
| Task start and end dates | 3/13 – 3/15 |

|  |  |
| --- | --- |
| Task name | Bingo JSP page |
| Task description | Change the HTML bingo table to JSP style by using the data model |
| Task start and end dates | 3/15 – 3/28 |

|  |  |
| --- | --- |
| Task name | Website template |
| Task description | Create a general website template for myIPFW Advisor |
| Task start and end dates | 3/29 – 4/4 |

|  |  |
| --- | --- |
| Task name | Login page |
| Task description | Added a servlet filter to handle LDAP login for security |
| Task start and end dates | 3/29 – 4/4 |

|  |  |
| --- | --- |
| Task name | Bingo sheet servlet |
| Task description | Create bingo sheet servlet to generate the sheet form the data model |
| Task start and end dates | 3/29 – 4/4 |

|  |  |
| --- | --- |
| Task name | Updated web design |
| Task description | 1. Change color scheme and style to better match IPFW’s style2. Create some tabs on top to switch between different areas |
| Task start and end dates | 3/29 – 4/4 |

|  |  |
| --- | --- |
| Task name | Display iCal calendars |
| Task description | Added a calendar to display the iCal calendars from the scheduler |
| Task start and end dates | 3/29 – 4/4 |

|  |  |
| --- | --- |
| Task name | Create editable bingo sheet |
| Task description | When advisor login, bingo table could be editable and the admin tab will show up. However, when student login, bingo table could not be editable and the admin tab will not show up.  |
| Task start and end dates | 4/5 – 4/18 |

3. You completed 11 SWEBOK reviews and 9 PMBOK reviews by now that outline software engineering and project management body of knowledge. Identify at least three Knowledge Areas (KA) from SWEBOK or PMBOK that pertain to your team roles this semester.

a. Describe the KA with figure [1 page] (You may use your SWEBOK/PMBOK reviews here)

Review Software Requirements (SWEBOK KA-1)



The software requirements knowledge area of SWEBOK deals with the elicitation, analysis, and validation of software requirements. A software requirement is a property that must be exhibited in order to solve some real-world problem. The knowledge area is further broken down into the following sub-topics: software requirements fundamentals, requirements process, requirements elicitation, requirements analysis, requirements specification, requirements validation, and practical considerations.

Since software requirements help to dictate what needs to be completed for a software engineering project to be considered a success, we dealt with this knowledge area a lot with our project. First, we had to solicit our clients for a set of features that they needed in the final product. This gave us a rough set of functional requirements. From there, we had to analyze each of the requirements to discover any possible resulting requirements. For example, the client requested that the system be able to process X number of queries per minute. As a result, the system gained the functional requirements of having to be deployable to a server that could handle those queries at that rate. Furthermore, the system had to be able to communicate to the different parts quickly enough to transfer the necessary information in an efficient manner. Once we had a more formal set of requirements, we had to verify that the requirements were accurate in relation to the original set of requested features from the client. To do this, we analyzed each requirement, along with any of its sub-requirements, to see if it met the corresponding client requested feature. We then made adjustments and additions to the set of requirements as needed to ensure that all of the feature requests were met. Finally, we met with our client to share our findings. After receiving positive feedback from the client that our findings met their requests, we were able to start the project design process.

Review Software Design (SWEBOK KA-2)



The software design knowledge area deals with the design of software systems. It can be viewed as a process in which software requirements are analyzed and translated into a description of the software system’s internal structure. This description can then be used during later software engineering life cycle activities in which it is translated into the actual software system. The design itself is defined as both ‘the process of defining the architecture, components, interfaces, and other characteristics of a system or component’ and the result of the aforementioned process. The knowledge area is further broken down into the following sub-topics: software design fundamentals, key issues in software design, software structure and architecture, software design quality analysis, software design notations, and software design strategies and methods.

As with any other software engineering project, part of our project’s process was to analyze the software requirements we came up with in the earlier engineering life cycle phases and translate them into a design of the overall software system. To begin, we took the final set of software requirements we had agreed upon earlier and analyzed each of them as well as any sub-requirements. The purpose of this analysis was to extract requirements for our design. For instance, if we had a software requirement that the system should store information about each driver’s insurance information, and then we could extract that the final system indeed needed to have a place to store such information. From there, we expanded on the extracted design hints in order to come up with a general outline of how we would handle each design feature. While this was generally implementation-agnostic, we did have to come up with a plan on how to meet each requirement. We continued to build upon each requirement’s design until we had met all of the software requirements. The end result was a combination of textual and graphical design requirements and descriptions. The set of graphical requirements and descriptions came in the form of UML diagrams that described not only what information would be stored in each module, but also how each module would communicate with one-another.

Review Software Construction (SWEBOK KA-3)



The software construction knowledge area deals with the translating the software design into the software system. More precisely, construction is the creation of a working software system through a combination of programming, verification, testing, and debugging. The knowledge area is further broken down into the following sub-topics: software construction fundamentals, managing construction, and practical considerations.

Just as with software requirements and software design, this phase of the project took a great deal of time as it encompassed several different phases including coding, testing, and debugging. To begin, we analyzed the software design we came up with in during the design phase of the project. We took a look at both the textual and graphical representations of the design to make sure that we fully understood the layout of the software system as a whole. From there, we began coding the actual modules for the system. Since we were following a pseudo rapid application development process in our development, we tested each of the modules after every major build. For instance, after finishing each major portion of the database / persistence layer, we ran unit tests on data inserts, deletes, updates, etc. We then analyzed the results in tandem with working on the other modules. Once the results were analyzed, we revisited any problem areas to apply fixes. Once that set of fixes was complete, we retested the affected modules. We continued to follow this cycle of code-build-test-fix until each module was complete. We also ran other tests on the sub-systems of the project as each one became complete. For instance, once we had the backend system complete, we ran a series of unit tests as well as manual tests on the system to ensure that all data transactions met the ACID requirements as well as data was transferred accurately. Once these tests were complete, we analyze the results and applied fixes as needed. We then re-ran these tests to see if the fixes were successful as well as to test if any new problems arose. This same cycle was applied to the system as a whole once all major modules were in place.

b. Describe your roles and their relationship to the area [1 page]

**Software requirements:**

For software requirements I didn’t actually design the FR-DP chart, but I did, however, provide input on how the functional requirements would be translated into software requirements. For instance, since we needed to display the bingo sheet for a user based on data from Connor’s database then I knew we would need to design a bingo sheet display in a web format. We would also need a way to generate the HTML for the display. This would mean we would need to implement a JDBC bridge to the database, Java objects to store the student’s information, and JSP to display the information to the student.

Some of the other requirements included a way to secure the web application. So I determined LDAP would be the most secure way to authenticate users. The LDAP method ended up providing us the IPFW ID for student as well so we could pull their data from ODS.

**Software design:**

Initially we just started programming without much concern for organization. Eventually we discovered it was too much to try to find code from different team members. Part of my role was to upload my code that I wrote to a central server that ran GIT. This allowed us to all work on the project at the same time, uploading changes to the server as we finished portions of the application, and then we could all get to the most up-to-date version of the code.

Since I was responsible for GUI for the application, a large portion of my role was to write a series of Java beans that would allow us to encapsulate the different data used to generate the display. This entailed reorganizing the data and logic to better separate the two. Eventually we settled on an MVC architecture where the model for the data is controlled by a series of servlets and displayed in views (JSP pages).

**Software construction:**

The construction area is basically writing the code and designing the layout of the web application interface. We have gone through several versions with increasing complexity but better layout each time. My general step is to first create rough outline of the code. I then go back and update the code to add missing features and start testing it. Finally, I integrate it into the existing code either through existing APIs or I alter the existing APIs to support the new code. Finally, I upload the changes to the GIT server for everyone else to use.

c. Provide diagrams/screenshots of any work output related to your area [1-3 pages] with captions.



<Figure1: Usecase Specification for user authentication>



 

 

<Figure2: User login page>



<Figure3: Display the bingo table>





<Figure4: View calendar for scheduler

d. Describe how you produced the work output in 3c [1-3 pages] (1 page per diagram)

**<Figure1 caption>**

User login page provides a user to put their username and password to the server to establish a session. The username and password is checked against the LDAP server. If the LDAP bind is successful then the credentials are considered valid and the session is created for the user.

Flow of Events

* Basic Flow

1. The user opens the MyIPFW Advisor web page.

2. The server asks user’s username and password.

3. User puts their username and password.

4. The server checks the username and password against the LDAP server.

5. If the LDAP bind is successful then the credentials are considered valid and the session is created for the user.

6. User can access their session.

* Alternative Flows

1. Invalid Credentials Provided

The provided username and password did not match with a valid, enabled, and unlocked account in the LDAP server.

1.1 Invalid Credentials Provided Subflow

No session is created for the user. The user is returned to the login page and shown an error message telling them their credentials are invalid and they should try again.

1. The user opens the MyIPFW Advisor web page.

2. The server asks user’s username and password.

3. User puts their username and password.

4. The server checks the username and password against the LDAP server.

5. If the LDAP bind is unsuccessful then the credentials are considered invalid and no session is created for the user.

6. User is redirected to the login page to try again and an error message is shown to notify the user their credentials were invalid.

2. LDAP Server Unavailable

The LDAP server could not be contacted to attempt a bind.

2.1 LDAP Server Unavailable Subflow

1. The user opens the MyIPFW Advisor web page.

2. The server asks user’s username and password.

3. User puts their username and password.

4. The server checks the username and password against the LDAP server.

5. If the LDAP bind is unsuccessful because the server could not be contacted then no session is created for the user.

**<Figure2 caption>**

An LDAP server is a server that organizes users into groups and stores details for each user including their real name, their employee or student ID number, their email address, etc. It also stores the user’s password. For example, if a user wants to access a web page then the web server would prompt the user for their user name and password and then check if they are valid. The web server could also check if the user was a member of a particular group. This way access to the web page could be given to all the users in a particular group (i.e. all CS professors can access a web page showing the grades for students in the CS department). So basically an LDAP server can be used to (1) lookup information about users and (2) validate user name and password.

1. Searching for entries in the LDAP server

You can connect to the IPFW LDAP server anonymously to lookup information about users. LDAP servers support a standard query language to search for entries, similar to how databases support SQL queries to find records.

The first thing to setup for the query is the scope. The scope is which part of the LDAP server to search. There are 2 major options: sub-tree or level. A sub-tree search looks through the entire server starting at the root location. A level search looks at just the current group.

For instance, if the root is “ou=Engineering,o=IPFW” then a sub-tree search would look in “ou=Engineering,o=IPFW” as well as “ou=ElectricalEngineering,ou=Engineering,o=IPFW” and “ou=MechanicalEngineering,ou=Engineering,o=IPFW” while a level search would only look in the “ou=Engineering,o=IPFW” group.

To search for an entry you use a LDAP query. In Java you can use the LDAP library’s search function:

NamingEnumeration<SearchResult> results = ctx.search("o=IPFW", "cn=" + userID, constraints);

The first parameter is the root location, the second parameter is the search parameters (i.e. cn=USERID to search for all entries where the cn=USERID), and the third parameter is the set of rules to follow when searching (i.e. search scope).

One important thing to remember is that while you can connect to the IPFW LDAP server anonymously, IPFW does protect the server from the external Internet. This means that you must either be on the internal IPFW network or you must connect to the internal IPFW network through the VPN server.

2. Validating usernames and passwords

While LDAP servers do store the username and password for users, you can’t just query the server for a user’s password. The LDAP server keeps this information protected to prevent hackers from stealing users’ passwords. To validate a user’s username and password you need to try to bind (or login) to the LDAP server using the user’s credentials. When you try to login, the LDAP server will compare the username and password against its internal database and then either allow the login if the username and password match a record, or reject the login if they don’t match a record.

Another thing to remember is that you can’t just login to the LDAP server using the user’s username and password. Instead, you need to provide the user’s distinguished name and the password. You can lookup the user’s distinguished name from the LDAP server in an anonymous connection and then try to bind the server using the user’s distinguished name and password.

**<Figure3 caption>**

To begin I looked at the original paper bingo sheet to get an idea of the design I would use. I then discussed the design with Professor Sedlmeyer and we decided to use a tabbed interface to display the different areas.

After we decided on the overall design, I created a bingo sheet using static HTML to display some sample idea.

Since we needed to be able to display the bingo sheet for different students based off of information in Connor’s database, I started researching ways of generating the HTML dynamically, based off of the data from the database.

The first way I did it was to generate the HTML as a long string and return it from a servlet to be displayed in the web browser. While the web page looked good, it was very difficult to read the code and was not very easy to make design changes.

I then read about a different way to combine Java and HTML called JSP. JSP allowed me to take the original HTML I created for the mockup of the bingo sheet and then substitute the variable data with data from Java objects. These Java objects where generated from data from Connor’s database. This allowed me to display the bingo sheet for any student in the database.

Since the web application consists of more than just the bingo sheet display, we needed a way to display content from other parts of the web application. At first I experimented with template engines, but I found them to be too much for such a simple application. The jQuery plugin I used to organize the bingo sheet into tabs worked well for this. The plugin can load HTML over AJAX and display it in the tab. This means I could provide a URL to the tab plugin and it would load HTML from the different servlets. That way we could show the content for calendar and admin pages as well.

**<Figure4 caption>**

One of the features discussed was providing a way for the student to view their proposed schedule in a graphical way, similar to the view provided in my.ipfw.edu. To do this, I researched HTML tables and ways to generate the different days of the month. I created a basic mockup of the month calendar, but ultimately it was going to be too much for a unnecessary requirement.

When I was researching jQuery plugins for the tabbed interface I remembered seeing a plugin for generating calendars from JSON data. The plugin was called FullCalendar and proved to be exactly what we needed. I was able to extract a list of locations, times, and class names from the iCal file and generate the JSON to be inserted into the JSP output.

The overall flow was:

Read the iCAL file

Extract list of meeting times, locations, and course names

Generate a long string that provides the information from the iCal file in a JSON-formatted string.

Pass this string to the FullCalendar plugin

The plugin parses the JSON data to create an array of calendar events

The events are displayed as blocks on the calendar

I also started working on a data model to get data from the ODS views. These views provide times for classes for all courses. With that I was able to create a method to take a list of course reference numbers and it would return a list of meeting times for those courses.

These events could then be translated into JSON and passed to the JSP and plugin the same way as the iCal was. This way we can take the output from the SugarCSP and display them on the calendar just the same way as my.ipfw.edu does.

Now I am working on a way to print the calendar without printing the other portions of the page. The only way I have found is to use jQuery to isolate the div containing the calendar and set the CSS for all other content to display:none. I can then use plain JavaScript to open a print dialog to print just the calendar and then display the other content after the print is complete.

4. Attach:

All your individual progress reports for each week of the semester already submitted in table format to your project manager, who routinely submitted to the program manager every Wednesday all teamwork completed for the week.  Hence, for each week, simply attach your individual progress reports that provide information on your role activity for each week.  These weekly reports should be assembled into 1 individual table separated by rows of weeks, which include at least:

a. Tasks completed for each week.

b. Tasks planned for the following week.

|  |  |  |  |
| --- | --- | --- | --- |
| Week # | Post Date | Tasks completed for each week | Tasks planned for the following week |
| 1 | 1/25 | 1. Review the SRS documentation and etc.2. Create my own Cmap. | 1. I am going to make a Usecase model about "User Authentication." |
| 2 | 2/1 | 1. Made an Usecase model about “User Authentication”2. Update our group Cmap by changing the overall structure and color of the items | 1. Revise Usecase diagram to Usecase specification2. Prepare three slides for the presentation |
| 3 | 2/8 | 1. Did 3 slides for presentation 2. Updated Use-Case specification by making steps. 3. Updated the group Cmap (Organizing and Color Coding) 4. Updated my Cmap | 1. Set up the same environment to start development2. Make a simple webpage to verify user’s credentials |
| 4 | 2/15 | 1. Set up Java EE and TOMCAT7, and investigated Dr. S. LDAP Test code.2. Made a simple webpage to verify the user's credentials. (Dr. S. assigned me on Tuesday when I met him individually) | 1. Imported the project files from Marat/Prof Wolfe into Eclipse |
| 5 | 2/22 | 1. Start looking the codes we have so far.2. Made documentation for authentication code.3. Updated Individual Cmap and Group Cmap.4. Did a PMBOK Review | 1.Start making BINGO Sheet (display Bingo sheet)2. Prepare three slides for next week group presentation.3. Update group cmap and individual cmap4. PMBOK Reviews |
| 6 | 2/29 | 1. Made three slides and prepare the presentation2. Updated group Cmap and individual Cmap3. Reviewed PMBOK KA-7 | 1. Prepare the exam 12. Create a bingo table in HTML |
| 7 | 3/14 | 1. Create a Bingo table using HTML (without Tab design)2. Create two classes in Java to generate Bingo contents3. Updated group Cmap | 1. Preparing midterm on Monday2. Study JSP, CSS, JQUERY this weekend3. Create Bingo web page including Tab design4. Update group Cmap if it is needed |
| 8 | 3/21 | 1. Display the B.S. CS Bingo Sheet in the web page by using HTML and CSS2. Studied HTML, CSS, and JQUERY to understand tab design3. Read Connor's Bingo API that I'm going to use | 1. Modify bingo.html and send it to Trent and Marat 2. Study JSP and test some simple codes by using Bingo data model3. Generate a rough Bingo web page by using JSP4. Update group Cmap if it is needed |
| 9 | 3/28 | 1. Studied JSP and test some simple codes by using Bingo data model 2. Generated a rough Bingo web page by using JSP 3. Integrate with Connor's data model | 1. Make three slides for presentation2. Revise the rough displaybingo.jsp |
| 10 | 4/4 | 1.Updating web design: Change color scheme and style to better match IPFW’s style Create some tabs on top to switch between different areas2. Create a general website template for myIPFW Advisor.3. Bingo sheet servlet to generate the sheet form the data model4. Added a calendar to display the iCal calendars from the scheduler5. Added a servlet filter to handle LDAP login for security6.Finish PMBOK\_KA-7, KA-8, and KA-9 | 1. Create scheduler tab2. Change editable bingo table for advisor3. Redo exam 14. Do 3 issues out of 18 issues in our team |
| 11 | 4/11 | 1. Researching and designing different styles of editable bingo table for advisor2. Researched about ical4J3. Updated group Cmap and individual Cmap | 1. Look Connor’s bingosheet.java and compare with my source codes. And change my jsp codes for using his bingosheet.java2. Create scheduler tab |