A Case Study in the Design of a Restaurant Management System

Wesley Williams, Devon M. Simmonds
Department of Computer Science
University of North Carolina Wilmington
{waw5709, simmondsd}@uncw.edu

Abstract

Teaching software engineering at the undergraduate level is an exciting and challenging undertaking. Students come to software engineering with a variety of technical and soft skills which have to be strengthened, honed and channeled to produce desirable results. This paper reports on the development of a restaurant management system as part of a first course in software engineering. Results and lessons learned are presented.

1. Introduction

At the undergraduate level, software engineering is taught as both one-semester and two-semester course sequences. In either format, teams of students create software with a level of complexity commensurate with the course sequence format. A familiar strategy to aid the teaching and learning process is to have students develop software in application domains to which can relate or are familiar. Restaurant management is one such domain.

In the experience of the authors, some restaurant management software while having reasonable utility features geared towards tracking money and profits, seem to have a significant lack where intuitiveness and usability are concerned. Some restaurant management software has weak or non-existent security, and in some cases, functionality needed for day to day operations are only available to managers. This results in the undesirable situation where access to manager-level functions are granted to lower-level employees. These problems may have resulted from the designer’s lack of domain knowledge, or from a lack of user involvement in the design process.

This paper reports on the design of a restaurant management software created as a team project in an undergraduate software engineering course. The overall goal of the project was to create an application where both utility and usability are addressed. In particular, the software should be easy to navigate and use while providing basic restaurant management and security features. In addition, the software should grant each employee a level of access that is commensurate with the duties and responsibilities required in performing duties.

When deployed, the application will serve as a single working “point of management” system that acts as both a terminal for taking orders and a terminal for generating reports and managing changes to employee records.

2. Project Plan

The main focus of the project was to create a single working “point of management” restaurant system that acts as both a terminal for taking orders and a terminal for generating reports and making changes to employees or items on the menu. Project planning was done to define the scope of the project, assess risks, and estimate and schedule project activities and thereby lay the foundation for the execution, monitoring and control of the project.

2.1. Project Scope

2.2. The overall scope of the software is reflected in the use case diagram shown in Figure 1. Major software functions of the restaurant management system include managing orders, inventory, and employee records, and generating reports.

Figure 1: Use Case Diagram

Order management includes creating and deleting orders, adding and removing items from an order and closing orders. Orders should also be stored in the database to be
used to calculate total sales. Inventory management includes adding new products, deleting products and updating products and resources.

The software is responsible for a number of other functions. Our software must be able to add employees, edit their information, and remove employees from the employee database. Menu items must be added, edited, and deleted from the menu item database. Items that can be ordered must be able to be added and removed from an order. All employees must be able to clock in and clock out. Servers must be able to do what all employees do as well as take orders. Managers should be able to do what all employees do and be able to edit item and employee information and generate reports. Reports that should be generated include sales reports showing sales by food category and the total sales from the start of the day. Orders should also be stored in the database to be used to calculate total sales. For a complete look at desired functionality see page six.

The non-functional requirements of the project included creating an intuitive, simple application that performs consistently. To achieve these goals the project should make menus effortlessly navigable and group user interface (UI) components in a manner that makes them easy to find. In addition, it was the opinion of the development team that giving each employee the appropriate level of access to resources was imperative for usability and security.

### 2.3. Risk Plan

Table 1 shows three risks that were identified for the project. The first of these risks is that of losing a member of the development team. This risk has a higher probability of occurring during the first weeks of the semester and diminishes as the semester progresses. The other two risks are more likely to occur and may occur at any time. As such, risk prevention strategies were developed and enacted. For example, soft copies of all documents were stored by all members of the development team to mitigate against loss of project artifacts.

<table>
<thead>
<tr>
<th>Risk</th>
<th>Prob.</th>
<th>Impact</th>
<th>Priority</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Losing a project member</td>
<td>2</td>
<td>9</td>
<td>10</td>
<td>Make sure every team member has updated copies of work done by all team members.</td>
</tr>
<tr>
<td>Hardware breaking down</td>
<td>4</td>
<td>2</td>
<td>10</td>
<td>Make sure every team member has updated copies of work done by all team members.</td>
</tr>
<tr>
<td>Task takes longer than expected</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>Trey will be free for helping other members once the initial setup of the database is complete.</td>
</tr>
</tbody>
</table>

### 2.4. Project Estimates and Schedule

Estimates for time and resources were calculated based on a developed work breakdown structure (WBS) for the project. The first two layers of the WBS is shown in Figure 2. Based on the WBS, tasks were assigned to members of the development team as shown in Table 2. The table identifies the persons who had primary responsibility for the development of a specific artifact or the management of a project task. However, all the members were given opportunity to modify and perfect artifacts and tasks assigned to other team members.

A schedule of tasks for the project was developed. The result is shown in Figure 3. The chart shows that implementation, testing and software design were the major project tasks, with implementation and testing consuming slightly more time than design.
The project team was compromised of a group of individuals with diverse programming and technical backgrounds. Most had experience programming in JAVA and .NET and most team members had a good grasp on the problem domain as many of them had worked in the restaurant industry at some point in their lives. Management was partially done via electronic means such as email and phone calls since many of the members have different and often conflicting schedules. The project manager scheduled one meeting per week for face-to-face communication and to verify the project status gain consensus and review completed individual tasks.

Table 2: Responsibility Matrix

<table>
<thead>
<tr>
<th>ID</th>
<th>Task Name</th>
<th>Start</th>
<th>Finish</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>User Registration and Project Selection Report Due</td>
<td>1/7/2009</td>
<td>1/15/2009</td>
<td>1.4w</td>
</tr>
<tr>
<td>2</td>
<td>Requirements Engineering Report Due</td>
<td>1/16/2009</td>
<td>2/9/2009</td>
<td>3.4w</td>
</tr>
<tr>
<td>3</td>
<td>Create Use Cases and Models</td>
<td>2/9/2009</td>
<td>3/9/2009</td>
<td>3.4w</td>
</tr>
<tr>
<td>5</td>
<td>Create Program Database</td>
<td>3/2/2009</td>
<td>3/9/2009</td>
<td>1.2w</td>
</tr>
<tr>
<td>7</td>
<td>Implement Sales</td>
<td>3/16/2009</td>
<td>3/29/2009</td>
<td>2.2w</td>
</tr>
<tr>
<td>8</td>
<td>Implement Manager Functions</td>
<td>3/16/2009</td>
<td>3/23/2009</td>
<td>1.2w</td>
</tr>
</tbody>
</table>

3. Software Design

While it is acknowledged that design is not always rational [8] and while we do not expect any silver bullet [3], in our experience, software design becomes better with practice and experience. For this project, the design of the software included both architectural and subsystem design. The architecture of software defines the major software subsystems and the dependencies and interrelationships among subsystems [1,2,4,6,7,10]. Architectural styles define a vocabulary for different classes of architectures [5,9]. Examples of well known architectural styles include pipe-and-filter, shared repository and event driven. In this course, each software development team was required to design the software using two architectural. The architectural styles chosen by the restaurant management team were the three-tiered layered architectural style and the shared-repository style.

Figure 3: Schedule of Activities

Figure 4: Shared Repository Architecture
The shared repository architectural style is shown in Figure 4. The design shows five major subsystems interacting with a shared data store. While this design is ideal for data driven applications that share a common database schema, it constraints the evolution of the database as well as the data formats on the individual subsystems. A layered architecture is shown in Figure 5.

Figure 5: Layered 3-Tier Architecture

3.1. User Interface Design

Because of the nature of our project, an intuitive graphical user interface was required. The user interface design in Figure 8 is the JAVA Swing equivalent of the earlier design. There are a few alterations that had to be made. The buttons on the far right side on each screen have been removed and put onto a single menu accessible after login. On the first screen visible to the user we have removed all buttons except the sign in button. The functionality that was originally on this particular screen has been moved to subsequent menus and screens. In addition, we have added the clock functionality to every screen in the program so that no matter which screen an employee is viewing, he or she will be able to keep track of time.

Figure 8: First user screen

The next screen (Figure 9) is where the employee can choose which tasks he or she wants to do. Options that appear grayed out are not available for the user that has logged in. Managers have all options, servers have all options except the management button, and finally normal employee are only granted access to system to log in and log out. All orders that are currently “owned” by this particular employee are listed in the list above the “add” and “edit” buttons. Below are three screenshots that show this (Figures 10, 11, & 12).

Figure 7: Edit item Sequence Diagram
Management menu functions include the ability to add and remove employees as well as items. Although our group did not implement report generation, report generation would be completed from the management menu. Below are screenshots of the management menu and the various submenus for editing employees and items.

The ordering screen allows a server or manager to create an order by selecting from four lists that represent the four types of food: beverage, appetizer, entrée, or dessert. When the item is selected, pressing the add button below the list adds the item to a fifth list which is the actual customer order. When the screen is exited, the information is stored in the database. All the screens are periodically updated with the current contents of the database. The figure below is the ordering screen.
3.2. Test Planning

Three goals were identified for the test plan: reliability, security, and usability. Reliability and security testing was accommodated by constructing test cases and comparing expected and actual results. Usability testing, however, is completely different. Usability testing would require some domain experts to use the software and perhaps even deploy the software in a restaurant environment. Due to the time constraints of the project we were unable to perform any formal usability testing using persons external to the development team.

Test cases were created to test adding, deleting, and editing both items and employees. Specifically these test cases make certain that employee and items are stored and retrieved from the database correctly. Test cases were also generated to perform boundary testing on how many entries could be successfully added or updated. In addition, test cases where created to verify the function of the compare class, which is used to validate input. The table of test cases listed on the following page shows what kinds of tests were performed on the “add new employee” function, the intended results, and the actual results.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Description</th>
<th>Intended result</th>
<th>Actual result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add new employee (0, Devon, Simmonds, 12.35, 3245)</td>
<td>Employee(1, Devon, Simmonds, 12.35, 3245) MANAGER</td>
<td>Employee(1, Devon, Simmonds, 12.35, 3245) MANAGER</td>
</tr>
<tr>
<td>2</td>
<td>Add new employee (0, John, Smith, 21.55, 1245)</td>
<td>Employee(2, John, Smith, 21.55, 1245) SERVER</td>
<td>Employee(2, John, Smith, 21.55, 1245) SERVER</td>
</tr>
<tr>
<td>3</td>
<td>Add new employee (0, Dan, Rather, 14.35, 378)</td>
<td>Employee(3, Dan, Rather, 14.35, 378) NORMAL</td>
<td>Employee(3, Dan, Rather, 14.35, 378) NORMAL</td>
</tr>
<tr>
<td>4</td>
<td>Add new employee(0, 45, 54, 23.12, 234)</td>
<td>Error</td>
<td>Employee(0, 45, 54, 23.12, 234) NORMAL</td>
</tr>
<tr>
<td>5</td>
<td>Add new employee(0, Hugh, Laurie, - 13.45, 546)</td>
<td>Error</td>
<td>Error</td>
</tr>
<tr>
<td>6</td>
<td>Add new employee(0, William,</td>
<td>Error</td>
<td>Error</td>
</tr>
</tbody>
</table>

Table 3: Sample Test Cases

4. Discussion and Lessons Learned

Students had some difficulty translating the architectures into design. As Parnas has noted [8], design rationality evolves as designers gain greater visibility into the artifact being designed and greater experience. This is especially true of novice designers who have little grasp of core design principles. One way to address this problem for institutions with only one software engineering undergraduate course is to introduce architectural concepts in the programming courses that precede software engineering. A second approach would be to introduce architectures very early in the software engineering course using problems and solutions from the earlier courses.

Students reported that their biggest challenge was time constraints. Implementation takes an extraordinary amount of time and a large amount of coordination. Scheduling project meetings around individual group member’s schedules creates many difficulties. Many of the group members were unable to devote the amount of focus that the implementation stage required. Both the former and the latter problem may be more of an issue in the academic environment where priorities of the different group members are skewed in a variety of directions.

Another issue that cropped up was knowledge of the Java programming language. At least two of the four group members were unfamiliar with Java’s Swing API, which is Java’s primary user interface package.

The team identified the responsibility matrix as the management construct most useful for managing the project, coordinating activities and tracking continuous improvement and overall success. All team members were assigned tasks, and everyone was held accountable for the completion of their assigned tasks. The responsibility matrix also allowed the team to track tasks that need to be done.
References


