|  |
| --- |
| Mercer University |
| Project 3 |
| SSE 656 - Object Oriented Project Methods |
|  |
| **Tanya Do, Grey Newell, and John Robison** |
| **12/5/2014** |

|  |
| --- |
|  |

Contents

[Introduction 4](#_Toc405486036)

[Problem Statement 5](#_Toc405486037)

[Initial Meeting With the Customer 5](#_Toc405486038)

[Commonality and Variability Analysis 5](#_Toc405486039)

[Features and Requirements 5](#_Toc405486040)

[Domain Analysis 6](#_Toc405486041)

[Use Cases and Use Case Diagrams 6](#_Toc405486042)

[Problem Submission Use Case 7](#_Toc405486043)

[Problem Judging Use Case 8](#_Toc405486044)

[Creating a Competition Use Case 8](#_Toc405486045)

[Use Case Diagram 9](#_Toc405486046)

[Technologies Used 10](#_Toc405486047)

[Architectural Analysis 13](#_Toc405486048)

[The Three Q's of Architecture 13](#_Toc405486049)

[Risk Analysis and Reduction 13](#_Toc405486050)

[Additional Discussion with the Customer 13](#_Toc405486051)

[Test Cases Used 13](#_Toc405486052)

[Development of the System 13](#_Toc405486053)

[Development of the Admin Portal Feature 13](#_Toc405486054)

[Development of the Problem Submission Use Case 14](#_Toc405486055)

[Development of the Problem Judging Use Case 14](#_Toc405486056)

[Test Results and Quality Analysis 14](#_Toc405486057)

[Conclusion 14](#_Toc405486058)

**Table of Figures**

[Figure 1: Use Case Diagram 10](#_Toc405486059)

[Figure 2: MVC Interaction Diagram 11](#_Toc405486060)

[Figure 3: A Visual Representation of Loose Coupling 12](#_Toc405486061)

[Figure 4: A Visual Representation of Strong Coupling 12](#_Toc405486062)

**Table of Tables**

# Introduction

# Problem Statement

# Initial Meeting With the Customer

# Commonality and Variability Analysis

# Features and Requirements

After meeting with the customer and performing a commonality and variability analysis on the system, our team derived a list of the features that would define the system as well as the requirements that would compose each of these features. These features are listed below:

* User Accounts
  + Teams
  + Judges
  + Admins
* Creating a Contest
  + Creating Problems
  + Adding Teams
  + Setting the Usable Languages
  + Setting the Start and End Times
* Viewing the Scoreboard
* Submitting a Solution
  + Uploading Source Code
  + Selecting the Used Language
* Judging a Submission
  + Claiming a Submission
  + Downloading the Judging Packet
  + Choosing the Appropriate Result
    - Correct
    - Wrong Output
    - Compile Time Error
    - Runtime Error
    - Presentation Error
* Reviewing Judged Submissions

The User Accounts will be responsible for allowing users to have different experiences with the system based on what their role in the competition is. These accounts will limit each user's operations and responsibilities within the system so that they are not able to overstep their limits. These responsibilities are as follows:

* Team Accounts will be the most common user to the system. They will be responsible for allowing participants to be able to submit solutions to problems and review their judged solutions. The submissions will consist of the user's source code which will be uploaded to the server as well as the programming language that was used in the submission.
* Judge Accounts will be responsible for allowing Team Accounts to have correct (or incorrect) submissions. They will be able to claim a submission and then judge it based on the output from the submitted source code. To allow a Judge the ability to judge submissions, the system will provide the Judge with a judging packet that consists of: the submitted source code, the judging input, and the judging output. Using these files, the judge will be able to run the source code with the judge input and compare this output with the provided judge output. The possible reviews that a Judge can give a submission are: Correct, Wrong Output, Compile Time Error, Runtime Error, and Presentation Error.
* Admin Accounts will be responsible for setting up and managing competitions. They will be able to add users (such as Teams and Judges), problems, and usable languages. The Admin will also be able to create a competition using this information as well as set the start and end times of these competitions.
* All users will be able to view the scoreboard that will show the ranking of the teams based on their number of correct submissions and penalty points accrued.

# Domain Analysis

# Use Cases and Use Case Diagrams

To ensure that our team has a more complete understanding of how the system should work, use cases were derived for the major interactions that users can have with the system. These interactions include:

* Submitting problems
* Judging problems
* Creating contests
* Creating problems
* Creating users

Our design team has determined that there will be three types of users for this system: Admins, Teams, and Judges. A use case of the main path was created for each of these types of users.

An Admin will be the user that sets up the competitions as well as creating user accounts for the Teams and Judges. The Admins will be able to specify all of the aspects of a competition such as: the Problems used in the competition, the Users participating in the competition, and the Languages that the Users will be able to submit their source code in. The Admin will also decide the start and end times of each competition.

Teams will be the main users of the system. They will be the users who are solving and submitting answers to the problems posted by the Admins. They will be able to decide which problem they want to solve and in what language they want from the lists provided by the Admin. After they have submitted their solutions, they will be able to review the judging of their solution and resubmit if necessary.

The final user group is the Judges. The Judges will be responsible for reviewing the Teams' submissions. To review these submissions, they will be proved the submitted source code as well as judging inputs and outputs. The judge will run the source code with the provided inputs and then compare the produced output to the provided output. Based on this comparison, the Judge will select the most appropriate response (i.e. "Correct Solution" or "Wrong Output").

The following subsections contain these use cases as well as the use case diagram that shows all possible interactions that a user can have.

## Problem Submission Use Case

1. The user navigates to the web URL associated with the judge application
2. The user logs in to the judge application with a team's credentials
3. The user views the scoreboard and selects a problem to solve
4. The user clicks the "Team" link
5. The user selects the solved problem from the drop down, selects the language used, and uploads the source file
6. The user waits for the problem to be judged
7. Once the problem is judged, the user returns to the Team page and views the judged problem to see the judge's comments

## Problem Judging Use Case

1. The user navigates to the web URL associated with the judge application
2. The user logs in to the judge application with a judge's credentials
3. The user clicks the "Judge" link
4. The user claims a submitted problem
5. The user downloads the zip file that contains the submitted source code, the input file, and the output file
6. The user runs the source code with the input file as the input to the system
7. The user compares the given output to the generated output
8. Based on this comparison, the user selects a judging option for the claimed problem

## Creating a Competition Use Case

1. The user navigates to the web URL associated with the judge application
2. The user logs in to the judge application with an Admin's credentials
3. The user clicks the "Admin" link
4. The user clicks the "Users" link
5. The user clicks the "New User" button
6. The user enters information for the new team and clicks the "Create" button
7. The user clicks the "Problems" link
8. The user clicks the "New Problem" link
9. The user enters a Problem name and provides the judge input and output and clicks the "Create" button
10. The user clicks the "Languages" button
11. The user clicks the "New Language" button
12. The user enters the language of their choosing and clicks the "Create" button
13. The user clicks the "Contests" link
14. The user clicks the "New Contest" button
15. The user selects the added team and problem, and selects start and end times
16. The user click the "Create" button

## Use Case Diagram



Figure 1: Use Case Diagram

# Technologies Used

This project employed the use of the ASP.NET MVC (Model-View-Controller) framework. The framework streamlines adoption of the MVC design pattern which encourages designs to be loosely coupled through a separation of concerns, and also enforces MVC related standards in our code. This separation is achieved by using the three main components: Models, Views, and Controllers. Models define the schema by which system data is stored and interacted with. Views are responsible for constructing the façade the user interacts with and displaying data from the appropriate Models in a human-readable fashion. Controllers are the “under-the-hood” components which respond to user inputs by updating the models and views accordingly. Below in Figure 2 is a diagram that visualizes the interaction between these components.



Figure : MVC Interaction Diagram

The ASP.NET framework builds on the MVC design pattern by relying on a "convention over configuration" approach which reduces the amount of code required to implement the design pattern for a project. This approach enforces certain facets of the design pattern, such that items are placed in the correct directories and named according to the MVC Schema; i.e. Controllers must be in the Controller folder and its name will end with "Controller.” Not only does this convention based approach reduce code requirements, it also aids in overall readability of the system.

Due to the loose coupling offered by the MVC design pattern, our team chose ASP.NET MVC as the framework for this system because it allowed for code reuse and distributed development in parallel. Since the models, views, and controllers are separated, they can easily be reused within another application, individually or as a whole. An example of this type of code reuse would be if the customer requested a mobile application. Model and Controller classes could be reused as-is for the backend, and all developers have to do is define new views for the mobile platform. This separation also allows components to be designed and built separately. One developer can build the store user interface in the views while another developer designs the underlying business logic in the models concurrently.

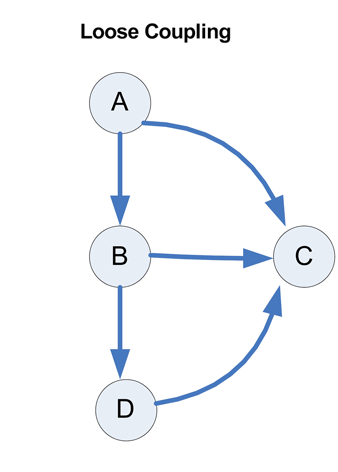


Figure : A Visual Representation of Loose Coupling

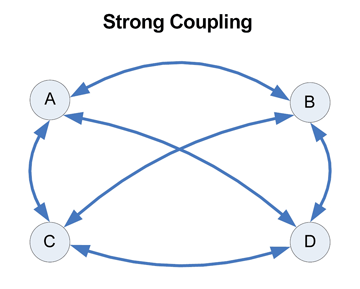


Figure : A Visual Representation of Strong Coupling

In addition to the aforementioned benefits of MVC's separation of concerns listed above, our team decided to develop with the ASP.NET MVC Framework for its use of .NET languages (in particular C# and Visual Basic) which we are already familiar with, and due to the success of our previous project employing this framework. In addition, the Razor view engine was easy to work with and teach to our newest developer, who actually became proficient with the engine quickly enough to implement most of our Views. C#, our .NET language of choice, is extremely powerful and easy to develop with due to its number of available libraries and the use of Microsoft Visual Studio’s Intellisense utility. Intellisense can auto-fill code for the developer such as variable and method names, whether the variable is a member of a project or one of its associated dependencies. The Razor view engine is a combination of HTML and either C# or Visual Basic, which makes it very natural for .NET developers to write and learn. The use of C# in the markup allows for very powerful dynamic web pages. Razor also allows the use of layouts which enables developers to have a single Razor file act as a template for all other views. These layouts reduce duplicate code by encapsulating common view elements into the layout. The final advantage of Razor that will aid in development is the ability to use Intellisense to quickly write the markup and code contained in the Razor files.

# Architectural Analysis

## The Three Q's of Architecture

## Risk Analysis and Reduction

# Additional Discussion with the Customer

# Test Cases Used

# Development of the System

## Development of the Admin Portal Feature

## Development of the Problem Submission Use Case

## Development of the Problem Judging Use Case

# Test Results and Quality Analysis

# Conclusion