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

|  |
| --- |
|  |

Contents

[Introduction 4](#_Toc405497567)

[Problem Statement 5](#_Toc405497568)

[Initial Meeting With the Customer 5](#_Toc405497569)

[Commonality and Variability Analysis 6](#_Toc405497570)

[Features and Requirements 7](#_Toc405497571)

[Domain Analysis 9](#_Toc405497572)

[Use Cases and Use Case Diagrams 9](#_Toc405497573)

[Problem Submission Use Case 10](#_Toc405497574)

[Problem Judging Use Case 10](#_Toc405497575)

[Creating a Competition Use Case 11](#_Toc405497576)

[Use Case Diagram 11](#_Toc405497577)

[Technologies Used 12](#_Toc405497578)

[Architectural Analysis 15](#_Toc405497579)

[The Three Q's of Architecture 15](#_Toc405497580)

[Risk Analysis and Reduction 15](#_Toc405497581)

[Additional Discussion with the Customer 15](#_Toc405497582)

[Test Cases Used 15](#_Toc405497583)

[Development of the System 15](#_Toc405497584)

[Development of the Admin Portal Feature 16](#_Toc405497585)

[Development of the Problem Submission Use Case 16](#_Toc405497586)

[Development of the Problem Judging Use Case 16](#_Toc405497587)

[Test Results and Quality Analysis 16](#_Toc405497588)

[Conclusion 16](#_Toc405497589)

[Challenges 17](#_Toc405497590)

[Goals for the Next Iteration 18](#_Toc405497591)

[Takeaways 18](#_Toc405497592)

**Table of Figures**

[Figure 1: Use Case Diagram 12](#_Toc405497593)

[Figure 2: MVC Interaction Diagram 13](#_Toc405497594)

[Figure 3: A Visual Representation of Loose Coupling 14](#_Toc405497595)

[Figure 4: A Visual Representation of Strong Coupling 14](#_Toc405497596)

**Table of Tables**

# Introduction

Our development team is Trident Developments. Our team members are Tanya Do, Alexander Newell, and John Robision.



In this paper, our development team will provide a problem statement about out the web-based application we created, covering the purpose of the application. We will provide the transcription of interviews we had with our customer, Mercer University’s Computer Science Department, where they provide in detail the requirements of the application and the context in which the application will be used, follow up concerns, and discussion of the progress of development.

Additionally, to maintain transparency between our team and the customer, we discuss our decisions and development process in detail. We have included our commonality and variability analysis. We’ve extracted information from the interview and compiled a features and requirements for the judging application. Domain analysis of the application is covered, as well as use case scenarios.

We choose to use the MVC (Model View Controller) design pattern for this application. And this paper, we discuss what MVC is, and why it was the best match for the application. We will also talk detail our architectural analysis and the three questions that determine architectural significance. We will then delve into coding development, dividing the discussion into parts of the application that were use-case-driven and parts that were feature-driven. And we’ll review our testing methods and the results.

# Problem Statement

Our development team was approached by the Mercer University Computer Science Department (MU CS Dept.), to create a tool to judge their annual programming competition. The MU CS Dept. is trying to expose and encourage more students to the field of computer science and programming. As a result of this initiative, the MU CS Dept. organizes and hosts an annual programming competition aim at college students.

We are building a web-based application that will facilitate the operations of a programming competition. A programming competition will have multiple teams submitting solution to problems, and a team of judges pulling the problems, and grading them.

# Initial Meeting With the Customer

Our development team received an email expressing interest in our services, from Mercer's CS Department. To clarify the requirements and purpose of the project, we scheduled a meeting to discuss the program judging application.



**Trident Developments:** Hello! We're so glad you considered our development team for your project. Could you tell us more about the programming competition you are organizing?

**Mercer CS Department**: Absolutely. We want to encourage more students to study Computer Science. And we thought it would be best to start before they start college. We primarily market our programming competition towards other college students. And we are thinking about expanding the number of participants for the competition. And so what we are using now would not support a larger number of teams participating.

**Tri Dev:** Okay, great. Could you please walk us through how the programming competition works?

**MU CS:** Yes. For the competition we have a set of programming problems that we hand out to the teams. The teams may solve the problem in the language of their choice: C#, python, java, C++, etc. So when they code up the solution to the programming problem, they have to log into the system with their team login. Go to the submission page, choose which language the solution is in, and then upload their solution file.

**Tri Dev:** Meanwhile, what does the Judge do?

**MU CS:** Well after the teams submits their solutions, the Judge is able to see the upload and the accompanying information about it. And then they run the solution on their own computers and compare the output with the correct output. If the solution is correct, the team’s score is posted on to the score board. If not, the Judge will give feedback on the type of error the solution produced. And post to the scoreboard that the number of times the team has attempted the problem so far.

**Tri Dev:** And who sets up the competition, such as start time and team logins?

**MU CS:** We would like an Administer role that does that. The admin should be able to set up the parameters of a competition. Such as how many programming problems there are, what the contest start date and time is, when the contest ends, and how many teams logins are needed. Additionally, they should be able to act as a Judge or a Participant.

**Tri Dev:** Okay, great. Thank you. We’ll get to work.

# Commonality and Variability Analysis

Based on this meeting with the customer, our team began analyzing other similar systems to see how they had approached the problem of providing a Programming Team Judge client to schools and competitions. The analyzed systems are: The judging client used at the Association Computing Machinery (ACM) International Collegiate Programming Competition (ICPC) hosted by Baylor University, the judging client used by the Consortium for Computing Sciences in Colleges (CCSC) Southeast Competition, and the open sourced judge client created by Chip Bell (<https://github.com/chipbell4/Judge>). Each of these systems has strengths and weaknesses that our team used as references in the design of our system.

There were features from each of the examined systems that would be useful in the design of our Judging Client. Other than the common scoreboard that each client had, some of these features were: the ability for teams to ask questions to the judges given by the CCSC client, the neatness of the web interfaces given by Chip Bell's client, and the layout of ACM's scoreboard. Our team really liked the idea of allowing teams to interact with the judges in a way that allowed common problems to be easily clarified. An intuitive and neat interface allows users to interact with the system more efficiently and reduce any unnecessary stress or strain on the users. Our team felt that the advantages of the ACM scoreboard was that it would highlight all correct submissions with special colors (such as gold for the first correct submission) as well as provide the time of the correct submission which would allow users to follow the trends of the competition and enhance strategies. However, since the team/judge interaction was not within the scope of the project given to us by our client they have requested that we wait until the second revision of this application to implement this feature. The other liked features will still be implemented because they do fall within the scope of the project.

While each system had features that were admired, they also had features that our design team did not want to implement in our Judge Client. These features include: The unintuitive and slow design of the CCSC client, and the lack of user interaction with Chip Bell and ACM's clients. While an intuitively designed user interface enhances a user's experience, a slow and unintuitive design will inhibit the user. Our team felt that the CCSC system was confusing to use and would often cause major slowdowns at various stages of the competition. Even though our client would like to wait to add the user/judge interactions, we still felt that the lack of this interaction was a downfall of both Chip Bell and ACM's systems.

# 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.

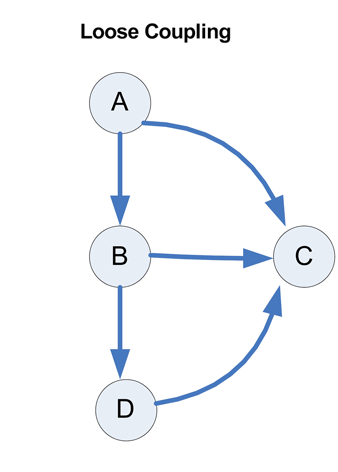


Figure : A Visual Representation of Loose Coupling

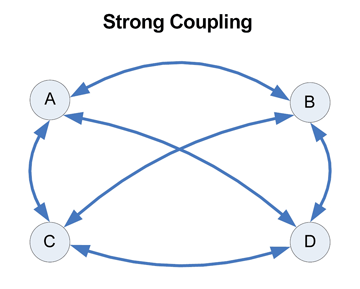


Figure : A Visual Representation of Strong Coupling

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

As a means of ensuring that the project is completed to the best of our abilities and within the time and budget constraints provided by our customer, our group performed an architectural analysis on the system to determine the possible bottlenecks and dependencies of the system. To establish the most architecturally significant portions of the system, our team looked at what are known as the "Three Q's of Architecture". Once the architectural significances were determined, our team used this knowledge to determine where the major risk areas were and devise a system that would reduce this risk as much as possible. These analyses are contained in the following subsections.

## The Three Q's of Architecture

A common method of determining the architectural significance of features and components in software development is to ask the Three Q's of Architecture for each major feature or component, which are as follows:

* Is it the essence of the system?
* What does the feature mean?
* How do you implement the feature?

Features are deemed more architecturally significant based on the answers that the developers have for each feature. The qualifications for significance are if the feature makes up the essence of the system, if the meaning of the feature is unclear to the developers, or if the developers are unsure of how to implement the feature. Features that meet more of the criteria are more architecturally significant than features that only meet one criterion or meet none of these criteria. Each of these three criteria is as important as the others, so these questions hold no additional weighs when determining architectural significance.

The features that have been determined to be the essence of the system are: user accounts, the admin portal, the ability to submit solutions, the ability to judge solutions, and a running scoreboard that shows the ranking of the teams. Without these features, the system would not meet the expectations of the customer and would not be considered a complete product. Users of the system will have to be able to log in to the system to interact with it. To allow these interactions to be specific, there will need to be three classes of users: Admins, Teams, and Judges. The admin portal will be used by Users in the role of Admin and is responsible for managing and creating all aspects of a competition: the problems for teams to solve, the users who can participate in the competitions, the usable languages, as well as the start and end times of the competitions. Having the ability to create and manage competitions is only useful if users are able to actively participate in the competitions. Because of this, it is important to establish a means for users in the Team role to submit their solution's source code to be judged. However, being able to submit problems is useless without having a mechanism that will assess the submission. To allow this to happen, a system will have to be implemented that allows a type of user who is a Judge to review a Team's submitted source code. To allow users to see their relative performance in a competition, the results of a Judge's review will be compiled into a scoreboard that is accessible by all user types.

Due to the nature of this system, there were no aspects of the project that our design team had uncertainties on the meaning of. This reduces the amount of risk inherent in the development of this system.

The features that our team is currently unsure of how to implement are the submission and judging of solutions. The fact that our team is unsure of how to implement these features makes them significant because they have the potential to cause blockages in development as well delays and budget problems. The component of the submission feature that our team is unsure of how to implement the ability to upload their source code and store it in a database in a way that can be later accessed to be judged. There will need to be a light weight solution for this problem that our team will decide on. After this problem is figured out, our team will then have to figure out how to use this submitted source code, the uploaded judging input, and the uploaded judging output to create a judging packet that will be a .zip file. This file, as well as the files that are contained within, will need to be created programmatically. Our team has no prior experience with handling these types of requirements, so extensive research will need to be conducted into these fields.

After performing this analysis using the Three Q's of Architecture, our team has determined that the most architecturally significant features of the system are: the user accounts, the admin portal, the ability to submit and judge problems, and an updatable scoreboard. The feature that was not considered architecturally significant was the user/ judge interaction. This feature was not considered significant for the following reasons:

* It is not a vital component for the system
* The feature had a clear and obvious meaning
* It has negligible time and resource requirements to complete
* It was deemed unimportant for this iteration by our client

## Risk Analysis and Reduction

After determining the architecturally significant components of the system, our team had to decide a planned order in which to develop these features in such a way that will reduce the risks inherent in development. Even though it did not contain the most qualifications from the Three Q's Analysis, the User Accounts will be developed first since all of the other features will build off of this. Having these accounts set up first will reduce any risks associated with integration of the other features. Also, since our team has prior experience with setting up a user account system, this will be a task that should cause little to no difficulties. This feature is set with the highest priority because there will need to be distinct user roles that will ensure that users can only access sections of the system that they are supposed to. An example of this is that a Team should not be able to access the Judging pages because they would then be able to artificially inflate their rankings by wrongfully judging their own work.

Following the development of the User Accounts, the Admin Portal will be developed because it is used to set up all aspects of the contests which the other features rely on. The development of this feature should not involve any delays or blocks because it is a type of feature that our team has dealt with before. Due to the other features' dependencies on this component, it was selected to be of a higher priority than the following features. Without this component, it will be a difficult process to create Users, Problems, Languages, and Competitions that the other features will need to be usable as well as to be tested for correctness.

After these two features have been developed, both the Solution Submission and Judging features will have an equal importance. However, our team has decided to deal with the Solution Submission first because it will be the more used feature of the two. The largest constriction of this feature is that our team will need to research how to store the source code in our database that will be able to later be accessed by the Judging feature. This task is of great risk to the development of this system since it is such a foreign task for our team to complete. After overcoming this hurdle, our team will develop the user interface that will allow a Team user to upload this file as well as specify which language was used. The task of designing and developing the user interface will be a trivial task compared to the task of storing the submitted source code, so it will be the last task completed of this feature.

After finishing this feature, our team will develop the system for Judging Submissions. The completion of this feature relies on the research that will be done for the Solution Submission feature. It will also require research into creating files as well as putting those files into a .zip folder programmatically. This is required because these files that comprise the .zip file will be used as the Judging packet when a Judge user claims a submission. This task is of great risk to the development of this system since it is such a foreign task for our team to complete. As with the Solution Submission feature, the next step in completing this feature is to design and develop the user interface that will allow a Judge user to claim a submission, download this judge packet, and then give a review of the submission based on the results of the judging packet. This will also be a trivial task compared to the task of creating the judging packet, so it will be handed last in the completion of this feature.

The final architecturally significant feature that our team will work on is the Scoreboard. The scoreboard will be an image that displays the current rankings of all Teams based on the number of problems that they submit correctly as well as the number of penalty points accrued. This task, while architecturally significant, poses little to no risk to the development of the system. Due to this lack of risk or dependency, it will be developed last. The scoreboard will need to be viewable from all types of users (Admins, Teams, and Judges) and will also need to update on its own. This scoreboard image will be programmatically generated whenever the home page of the service is accessed, which is a trivial task using built in C# libraries.

Using a set schedule such as this will greatly reduce risk in both the design and implementation stages of development. The ordering of the completion of these tasks is as follows: the creation of User Accounts, the development of an Admin Portal, the development of the Solution Submission process, the development of the Submission Judging process, and finally the implementation of the scoreboard feature. For the reasons listed above, our team believes that this ordering of tasks will mitigate many of the risks that are inherent in the development of this system and will help to ensure that the project remains on time and on budget. This risk mitigation plan will also allow our team to carefully plan for any bottlenecks or complications in the development process and be able to appropriately and efficiently deal with these concerns.

# Additional Discussion with the Customer

At Trident Developments, we pride ourselves on communication skills and transparency with our customers. After intensive analysis and planning, we decided to give Mercer’s Computer Science Department a status report on their judging application. So gave them call and a set up an in-person meeting.



**Trident Developments:** Hi! We’re glad you could make it. We wanted to sit down and give you an update on the development of your Judging application.

**Mercer University Computer Science Department:** Great! We’re glad to hear that.

**Tri Dev:**  During our commonality and variability analysis, we compared the features and requirements of your Judging application with existing ones.

The three we took a look at were the ones used by the ACM International Collegiate Programming Competition, Consortium for Computing Sciences in Colleges Southeast Competition, and the open source one created by Chip Bell.

**MU CS:** Okay, good. Those are the other judging applications that we took a look at, before coming to you guys.

**Tri Dev:** We found that the features that hindered the other Judging applications was implementation of an unintuitive design. And slow reaction time to user input.

**MU CS:** Right. What we saw with the CCSCSC’s Judging application that we really liked, was the feature that allowed participants to ask the judges questions about the problems they were solving.

**Tri Dev:**  We also see that as a positive, however, as we’ve discussed earlier, it would be best to wait for the next iteration of the application to implement that feature.

**MU CS:** We definitely understand. We did just want to iterate that it is a feature that we want in the future.

**Tri Dev:**  We also decided to use the Model-View-Controller (MVC) design pattern. ASP.NET has a framework that supports this pattern. This framework encourages the design of the system to be loosely coupled, via separation of responsibility.

The decoupling of the different parts of the system will allow for our team to work on multiple parts of the system simultaneously. This increases productivity and decreases development time, resulting in you receiving your application much quicker!

**MU CS:** That sounds great!

**Tri Dev:** Additionally, it will be much easier to make changes if you wanted to update the system. And therefore cheaper for you too.

If you wanted us to just go and change the way the application looked, we could easily and quickly do that with little or no disruption to you. The way MVC is set up, we the looks of the application should have no bearing on how the system works.

**MU CS:** Great. We definitely like the maintainability aspect of your plan.

**Tri Dev:** And in front of you are the use-cases and use-case diagrams we’ve compiled for you.

**MU CS:** Could you explain to us what they are?

**Tri Dev:** Yes. Definitely.

So in the individual use-case scenarios, we start off by picking a user and then picking an initial action they would do in the system and following it to completion. We repeat this until all possible actions have been covered.

**MU CS:** Great so what is this diagram then?

**Tri Dev:** This diagram is a use-case diagram. And it the same thing as the use-case scenarios. Except, this puts all of the scenarios into a graphical format. The graphical format is so you can see all of the use-cases at once. And also, how the different roles relate or do not relate to each other.

**MU CS:** Okay. The project plan looks great so far.

**Tri Dev:** Definitely feel free to let us know if you have question or anything.

**MU CS:** Thank you for the update. Please continue to keep us informed.

# Test Cases Used

# Development of the System

Along with the Test Driven Development methodology that our team implemented, we also used Feature Driven Development. While Test Driven Development deals with ensuring that the behavior of a system is as expected, Feature Driven Development deals with deciding logical sections to code a solution to the given problem. Feature Driven Development is the process of working on a specific piece of functionality one at a time. Feature Driven Development is most useful in systems that have many disconnected features or features that don't interact much. One advantage of this approach is that it allows developers to have working features very quickly which can be shown to customers as proof of work completed. Another advantage is that this approach ensures that all features are accounted for and are fully fleshed out. Feature Driven Development was used in the design of the Admin Portal, discussed in the following subsection.

Another development methodology we employed in bringing this project to successful completion is Use Case Driven Development. This process means selecting your features and components to develop according to a certain scenario from a use case. When a Use Case Driven development cycle is over, all features necessary to perform every scenario within a certain use case should be implemented (as much as necessary) and tested. Tackling development with this strategy is a more user-centric and less granular approach as compared to feature driven development, and is considered a “big-picture” approach by the authors of *HeadFirst OOA&D*. Developers often utilize this approach when the system has many scenarios and processes that comprise a single, cohesive user experience. The Problem Submission and Problem Judging portions of our system were developed in this manner.

## 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

In the course of conducting this project, the team displayed a knowledge of object oriented analysis and design by thoroughly executing an iteration of the OOAD lifecycle. Beginning with client interviews, the team conducted Commonality vs. Variability analysis of the anticipated product compared to other solutions to the same or a similar problem. Next, concrete lists of features and requirements of the solution were derived from the client interviews and CvsV analysis. Using the features and requirements garnered from the previous steps, the team developed and diagrammed use cases which described how users would interact with the system. The use case diagrams suggested how the internal workings of the system might be broken into components and subsystems, and therefore subsequently arranged. This implied structure was refined and codified during our Architectural Analysis, where the team applied The Three Q’s of Architecture to determine how each problem should be addressed. At this point, the team found it necessary to consult the client for verification and additional information, which included sharing with the client all analysis and design work products. The client approved of our design, and we moved forward by defining Test Cases which the system must satisfy. After the test cases were written and approved, the team began distributed parallel development of our web service using different development methodologies. Once development was considered feature-complete a final testing phase was conducted to ensure all unit and integration tests passed, all use cases were satisfied, and user acceptance testing was valid. After a brief quality review, the system was complete!

The way our project was conducted demonstrates the application of a mature understanding of the software development lifecycle as it applies to an object oriented system. In the end, we

### Challenges

One of the biggest challenges the team faced during development resulted from the **distributed team development environment**. While working on the section of the project we were individually responsible for, often it became necessary to interact with someone else’s portion of the system- which may or may not exist yet. Furthermore, when designing our own components we made decisions which heavily impacted the way other components should be built or the way that ours would be interacted with. To overcome this challenge, the team employed the techniques of Scaffolding, Programming by Contract, and Communication.

* **Scaffolding** is a technique developers can employ thanks to encapsulation. In a situation where the functionality being written or tested relies on others which have not yet been written or tested, developers can construct Scaffolding, or temporary boilerplate pieces of code which return the appropriate values the constructed system would. This allows developers to focus on their own area of expertise, as well as increasing separation between Unit and Integration tests. A unit test generally employs scaffolding to isolate only the functionality developed within that Unit, while an integration test will utilize the component itself. Scaffolding was almost totally replaced as development proceeded.
* **Programming by Contract is the practice of defining a standard for the way you expect users to interact with your code. Even if Tanya, for example, didn’t know how the Problem Judging subsystem would specifically use her Problem Submission subsystem, she had the ability to define a Contract by which all interaction took place. This meant that to use her code, you had to follow the standards for input/output, organization of code, and separation of responsibility in the way she had already defined it. This technique is useful because it allows a developer to assume objects, methods, and data within her system are populated in the way that she would expect them to be. For example, if Tanya programmed by the contract that no null variables or objects would be passed, she is allowed to assume everywhere in her system that nothing is null, saving valuable time checking input and exceptions.**
* **Communication** is perhaps the simplest and yet most important tool for developers to overcome the challenges associated with the distributed development environment. Developers utilized dynamic communication methods (like texts and phone calls) to resolve simple factual questions that might arise, such as “How are problem submissions stored in the database?” In conjunction with this, static communication methods (like a github repository) were established to communicate information such as standards, code, and responsibilities which would be hard to express otherwise. Whenever information was needed about a specific facet of the project, developers could access this information using either tool.

### Goals for the Next Iteration

The scope of the project covered in this report has been completed, but the product itself has further goals established by the developers and the client.

* **Communication between Teams and Judges:** The client was overjoyed to receive our completed system, but almost immediately turned around and requested this feature. They felt that while in their current use case (localized programming competitions) it was very simple to communicate between Judges and Teams, what if they decided to host an online competition? A facilitated communication method within our web app opens up many possibilities for distributed competitions, and also improves usability of the app as a whole.
* **Content Filtering:** In light of the new distributed use case the client would like implemented, some type of content filtering must be implemented in order to prevent inappropriate team names, malicious code submissions, and fraudulent entries. While in the current use case it is up to the host of the competition and his Admins to protect against these types of threats, in a distributed environment our framework will need to handle some of these functionalities for the Admin, and provide tools to assist in the others.
* **Automated build/run of submissions:** Currently, judges manually download and compile solution candidates. The judging process would be quicker and better if our application built the submissions automatically and ran them with the sample input given, allowing judges to spend all of their time examining the source code and output.

When the next design iteration begins, the team will embark on a similar process as the one this report describes. After the appropriate design and analysis has conducted, development will proceed, and hopefully the customer will be just as pleased with the next iteration as this one.

### Takeaways

As this project was meant to demonstrate a holistic understanding of the concepts presented within the text as well as how to apply them, the team came up with several Takeaways. These are bits of knowledge we gained from executing the project which either were not shared in the text, were at first counterintuitive, or are revelations personal in nature.

* **Existing design patterns are more useful the more skill and practice you have in applying them.** The original members of the team had already worked extensively with the Model-View-Controller pattern and its ASP.NET implementation in the previous project. The newest member of the team shared only a passing familiarity with the concepts of MVC, but was very quickly “brought up to speed” by the team. Several important ideas such as what to encapsulate where make perfect sense in theory, but specific functionality sometimes doesn’t intuitively fit into one of the three MVC components. It was only through experience that the whole team became skilled in working with our chosen design pattern.
* **Both distributed and centralized version control systems are useful; which is optimal depends on domain and preference.** Initially the team began development using Git, a distributed version control system. This allowed the team to develop quickly, utilizing feature branching to very quickly launch into their own portions of the project. Soon, development slowed to a crawl as integration errors began to occur, important overlapping functionalities began to malfunction, and system-specific errors began to occur. When the team switched to a centralized version control system, the one clean working copy was preserved all the rest of the way through development. Lesson learned? Each VCS has its area of application, and a skilled designer will be able to tell beforehand which one is likely the better choice.
* **Design is intrinsically both a predictive and reactive activity.** In the world of software development methodology, a war rages on between two schools of thought. One school of thought, Lean and Agile Methodologies, states that design should sport a purely iterative and reactive approach by identifying and conforming to customer requirement. Waterfall Methodologies on the other hand state that it is the designer’s job to anticipate these requirements and plan for them beforehand, preventing the need to adapt. Conducting this project the team has learned that in the real world, it takes a little bit of both strategies. Although a thorough design phase was conducted before the first line of code was committed, at several points in development small design changes were necessary- and not just due to customer requirements! When a particular piece of functionality doesn’t or won’t work in the way the designer had imagined, every intertwined piece must change accordingly. Thanks to our commitment to build a loosely coupled application based firmly in OOA&D principles, overall impact to the system and development time remained relatively small, and the final product delivered was more than satisfactory.

In conclusion, we hope this project has demonstrated our understanding of the material and how to use it in our development projects. It’s been a fantastic semester, and good luck to the rest of the class with the remainder of finals!