Honours Project Report

eSports Management Portal

By Sharad Parbhoo

Supervised by: Professor Sonia Berman and Gary Stewart

<table>
<thead>
<tr>
<th>Category</th>
<th>Min</th>
<th>Max</th>
<th>Chosen</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Requirements Analysis and Design</td>
<td>0</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>2 Theoretical Analysis</td>
<td>0</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>3 Experimental Design and Execution</td>
<td>0</td>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>4 System Development and Implementation</td>
<td>0</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>5 Results, Findings and Conclusion</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>6 Aim Formulation and Background Work</td>
<td>10</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>7 Quality of Report Writing and Presentation</td>
<td>10</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>8 Adherence to Project Proposal and Quality of Deliverables</td>
<td>10</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>9 Overall General Project Evaluation</td>
<td>0</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total Marks</strong></td>
<td>80</td>
<td></td>
<td>80</td>
</tr>
</tbody>
</table>

Department of Computer Science

2014

University of Cape Town
1. **Abstract**

eSports has become more popular around the world as gaming technology advances. People in Hong Kong can be competing over vast oceans with others in America or any part of the globe. Derivco, a software development company in Durban, South Africa, want to take advantage of eSports to build relationships with their partner officers scattered all around the world. This project aims to create an environment to host and organize eSports leagues and tournaments within Derivco. We have developed an online eSports management portal that caters for users in different time zones, making league and tournament organization easier to track and manage. The project is separated into three components, the player interface, the administrator interface and the mobile interface. The focus on this report is specifically on the design and implementation of the administrator interface.
## Table of Contents

1. **ABSTRACT**  
2. **TABLE OF CONTENTS**  
3. **INTRODUCTION**
   1.1. **eSPORTS**
   1.2. **DERIVCO**
   1.3. **PROBLEM OUTLINE**
   1.4. **PROPOSED SYSTEM AND DIVISION OF WORK**
   1.5. **HIGH-LEVEL REQUIREMENTS**
   1.6. **REPORT OUTLINE**
4. **BACKGROUND CHAPTER**
   2.1. **OVERVIEW**
   2.2. **INTRODUCTION**
      2.2.1. **THE RISING POPULARITY OF THE WEB**
      2.2.2. **AN EXPLANATION OF SOFTWARE ENGINEERING**
   2.3. **MOTIVATION FOR WEB ENGINEERING**
      2.3.1. **THE WEB CRISIS**
      2.3.2. **SIMPLE VS. ADVANCE WEB APPLICATIONS**
      2.3.3. **ASPECTS OF WEB DEVELOPMENT**
      2.3.4. **WEB ENGINEERING AS A SOLUTION**
      2.3.5. **AN AGILE APPROACH**
   2.4. **AGILE WEB ENGINEERING**
      2.4.1. **OVERVIEW OF AWE**
      2.4.2. **COMPONENTS OF AWE**
      2.4.3. **SUMMARY**
   2.5. **WEB SEMANTIC DEVELOPMENT METHOD**
      2.5.1. **OVERVIEW OF WSDM**
      2.5.2. **COMPONENTS OF WSDM**
      2.5.3. **SUMMARY**
   2.6. **A USER CENTERED DESIGN APPROACH**
      2.6.1. **OVERVIEW OF UCD**
      2.6.2. **COMPONENTS OF UCD**
      2.6.3. **SUMMARY**
   2.7. **COMPARISON OF METHODOLOGIES**
   2.8. **CONCLUSION**
5. **REQUIREMENTS ANALYSIS**
   3.1. **BUSINESS CASE SUMMARY**
   3.2. **REQUIREMENTS ELICITATION**
   3.3. **CORE CONCEPTS AND TERMINOLOGY**
   3.4. **FUNCTIONAL REQUIREMENTS**
      3.4.1. **ADMINISTRATOR REQUIREMENTS**
      3.4.2. **PLAYER REQUIREMENTS**
      3.4.3. **MOBILE REQUIREMENTS**
   3.5. **NON-FUNCTIONAL REQUIREMENTS**
      3.5.1. **MAINTAINABILITY**
      3.5.2. **DATA**
      3.5.3. **USABILITY**
3.5.4. Scalability 27
3.5.5. Performance 27
3.5.6. Security 27
3.6. Use Cases 28
  3.6.1. Create a User 29
  3.6.2. Create Tournament 29
  3.6.3. Schedule Knockout Tournament 29
  3.6.4. Schedule Round-robin Tournament 29
  3.6.5. Resolve Disputes 29

4. Esports Portal Planning and Design 30
  4.1. General Overview 30
  4.2. Agile Development Methodology 30
    4.2.1. AgileFANT 31
    4.2.2. Version Control 31
  4.3. Privilege Control 32
  4.4. Design Phase 32
    4.4.1. Model-View-Controller (MVC) Architecture 32
    4.4.2. Conceptual Models 33
    4.4.3. Entity Models 34
    4.4.4. Site-Map 37

5. Implementation Phase 38
  5.1. General Overview 38
  5.2. Sprint Cycle One 38
    5.2.1. Administrator Users and Account Management 38
    5.2.2. Leagues 39
    5.2.3. User Authentication 39
  5.3. Sprint Cycle Two 40
    5.3.1. Tournaments and Scheduling 41
  5.4. Sprint Cycle Three 43
    5.4.1. Core Tournament Scheduling 43
    5.4.2. Knockout Tournament Scheduling 46
    5.4.3. Round-robin Tournament Scheduling 48
    5.4.4. Dispute Resolution 51
  5.5. System Workflow 52

6. Testing Phase 53
  6.1. Functional Testing 53
    6.1.1. Scheduling Accuracy 53
    6.1.2. Dispute Resolution 54
  6.2. Non-Functional Testing 54
    6.2.1. Scalability, Performance and Load Testing 54
    6.2.2. Integration Testing 54
    6.2.3. Security Testing 55
  6.3. Exception Handling 55

7. Conclusion 56
  7.1. Summary 56
  7.2. Reflection 56
  7.3. Evaluation 57
  7.4. Future Work 57
7.4.1. Additional Social Information 57
7.4.2. Logging Scheduling Disputes 58
7.4.3. Customizable eSports News Feed 58
7.4.4. Tournament Type Variety 58

8. Bibliography 59

9. Appendix 61

9.1. A: Sprint Cycles Two and Three Overview 61
9.2. B: CalculateOverlap Method 62
9.3. C: GetOverlaps Method 63
9.4. D: Database Schema 64
1. Introduction

1.1. eSports
Electronic gaming has increased in popularity over the past decade paralleling the advances in gaming and console technology. In fact, it has moved from a purely entertainment means to a competitive multibillion dollar industry, bringing people of all nationalities together. It reaches millions of people from around the world and has expanded social networking borders to include even spectators, which has earned it its right as a new category in the sporting world called eSports.

eSports refers to electronic gaming competitions involving all genres of electronic games, from fighting to strategy to actual sport (IGN, 2013). It does not discriminate between gender or age, creed or colour, which has made it the perfect platform for Derivco to build new relations aligned with their recent expansion into the overseas space.

1.2. Derivco
Derivco is a software development company based in Durban, South Africa that specializes in Microsoft .NET development. They have recently acquired companies in different countries including Hong Kong, Australia, Isle of Mann and Estonia. These companies are also in the software development industry, and were naturally seen as competitors prior to acquisition. However, now that they have partnered up, Derivco aims to take advantage of eSports and use it to bridge the geographic spread of different branches and build a corporate culture.

To this end, Derivco is setting up eSports leagues to facilitate inter-branch competitions.

1.3. Problem Outline
Derivco wish to manage these leagues via a Web-based system. They currently have a limited system in place that was created by their in-house IT support team. While it has been sufficiently functional for their local needs, it does not cater for the new demands required by the recent acquisitions. However, the success of the current system has shown the idea has enough merit to warrant the development of a more effective system.

The new system will need to consider differing time zones of participants to appropriately schedule fixtures, support for multiple gaming titles and the capacity to host many more users.
1.4. **Proposed System and Division of Work**
The aim of this project is to develop a Web-application that can be used globally to manage eSports players and tournaments within Derivco and their partner offices.

The system has been divided into three components, the administrator interface, the player interface and the mobile interface. I am responsible for the administrator component, Connor Patrick is responsible for the player component and Hayoun Noh the mobile component.

The administrator and player interface had to be developed with the Microsoft .NET framework so that Derivco’s IT department can easily maintain it. Under the .NET framework, HTML and CSS were used for the front-end in conjunction with C# for the business logic and Microsoft SQL Server was used to connect the front-end to the back-end. The listed technology stack is included in the Microsoft Visual Studio 2013, which is our development environment of choice for .NET.

For the mobile component, an Android mobile application connects to the back-end of the system through custom Web-services.

1.5. **High-level Requirements**
The administrator needed to support the following features:

- Create users, leagues and tournaments
- Manage users, leagues and tournaments
- Schedule matches accounting for different time zones and player availability
- View and resolve disputes

The Administrator Subsystem described in this report thus forms the core of the project, with project partners’ Web and Mobile Player subsystems retrieving and altering some of the information managed by the Administrator.

Players join and leave leagues and tournaments, upload results, view upcoming fixtures, track information about leagues and tournaments being played, and log disputes if there is a discrepancy between the uploaded scores from each team.

The mobile application is a subset of the Player Web application due to its unique constraints such as its small screen size and limited memory. It includes the functionality to take pictures of results and upload them as proof.
1.6. **Report Outline**

This report will introduce the topic with a background chapter. The high-level requirements will be further analyzed in Chapter 3, and these requirements will be used in the planning and design phase in Chapter 4. Chapter 5 will examine the implementation phase, which discusses how we modeled classes and developed the logic behind controllers. Following that, the testing and evaluation of the system is analyzed, and lastly, the conclusions and future prospects are discussed in Chapter 7.

It must be noted that this is a software engineering project and as such there are no research questions. In this regard, Derivco indicated a strong preference for the work to be divided among team members according to functionality rather than according to front, middle and back-end layers, as a result of which the work division outlined here was adopted.
2. Background Chapter

2.1. Overview
As an introduction to the project, because this is my first time developing a Web application, I thought it would be relevant to research the methodologies specific to building Web applications. As such, I have analyzed three Web development methodologies, and compared them in this background chapter.

2.2. Introduction

2.2.1. The Rising Popularity of the Web
Over the past two decades, the Internet and the World Wide Web has grown in it's ubiquity, popularity and extent of use and is continuing to grow at an exponential rate, surpassing all other technological developments in history (Murugesan & Ginige, Web Engineering, 2001). This technology is used to enhance operations making interaction easier within society, and in industries ranging from travel to commerce to education. On the other hand, the advent of the Web connects society to a wealth of information as well as to each other, bridging geographical divides on a global scale.

The demands of Web applications have also increased significantly over the years (Murugesan & Ginige, 2006). In addition to this, recent advances in wireless technologies, smart phones and other portable computing devices have sparked a revolution in mobile Web applications (Murugesan & Ginige, 2006). A combination of the rise in Web use and its increasing application complexity has made the design, development, deployment and maintenance of Web application projects more complex and difficult to manage.

As our dependence on Web-based systems is increasing, their performance, reliability and quality become more important, which is where software engineering plays a critical role. There are aspects of software engineering that contribute to the sustainability of these Web development projects. Before exploring these aspects, a brief introduction to software engineering is required.

2.2.2. An Explanation of Software Engineering
Software engineering (SE) is an active field of research and has been since the late 1960's. It is defined as,

“The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software” - IEEE Standard Glossary of Software Engineering Terminology

The objective of SE is to provide a disciplined approach to software development to produce systems that
- Are reliable and robust,
- Address the problems they were developed to solve, and
- Are delivered on time and within budget.

New technologies are continuously emerging. Legacy technologies are constantly evolving. These two factors have a direct impact on the way software is developed, and thus the need for a sound software development methodology has become essential as criteria for successful software projects. These findings serve as the foundation for SE methodologies, which have recently been classified as either Monumental or Agile. Monumental process focuses largely on documentation prior to development whereas Agile methodologies are concerned more with software deliverables than document deliverables (McDonald & Welland, 2001). The principles of Agile development are discussed further in Chapter 7.
2.3. Motivation for Web Engineering

2.3.1. The Web Crisis
Web applications are being built with only the primary objective of the application in mind. Several other significant factors are not being considered such as the users’ needs, and issues relating to content management, website design, performance, security, maintenance and scalability, copyright and privacy.

San Murugesan, editor in chief of the *IEEE IT Professional*, describes some of the issues associated with the low quality Web systems *(Murugesan et al, 2000)*.
- Outdated data
- Irrelevant information
- Difficulty using website
- Slow Web page responses
- Website crashes
- Security breaches

2.3.2. Simple vs. Advance Web Applications
The ad hoc approach may have been appropriate for simple content presentation, for instance personal blogs. However, there has been a shift with the introduction of the Web 2.0 to more complex systems that contain text, images and other multimedia. Murugesan and his co-authors compare simple and complex Web systems in his paper *Web Engineering*. A summary can be seen in the table on the following page.

Table 1: A comparison between simple and advanced Web applications

<table>
<thead>
<tr>
<th>Characteristics of Simple and Advanced Web Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Simple Web-based Systems</strong></td>
</tr>
<tr>
<td>Primarily textual information</td>
</tr>
<tr>
<td>Static information content</td>
</tr>
<tr>
<td>Simple navigation</td>
</tr>
<tr>
<td>Limited usefulness</td>
</tr>
<tr>
<td>Performance not a major requirement</td>
</tr>
<tr>
<td>Easy to create</td>
</tr>
</tbody>
</table>
2.3.3. Aspects of Web Development
There is thus considerably more to Web development than visual design and user interface. It involves planning, Web architectures, system designs, testing and performance evaluation and frequent update and maintenance of the system as the requirements and usage grow.

Hence ad hoc development is not appropriate for complex Web systems. This approach could result in serious problems affecting the final system as per the customer specification, the performance, security, maintainability and scalability of the system and the schedule and budget of the project.

More importantly, faulty Web systems potentially impact business operations and transactions. Organizations that rely on the Web cannot afford frequent downtime or stale information. Inconsistent service frustrates users, costing the enterprise heavily in terms of financial loss, lost customers and ultimately, loss of reputation (Murugesan & Ginige, Web Engineering: Introduction and Perspectives, 2006).

Murugesan further explains that although we are faced with these challenges, most Web application development continues to be “ad hoc, chaotic, failure-prone, and unsatisfactory”. He goes on to say that this state of affairs could get worse because Web systems are becoming more complex and require more interaction, both from users and other automated systems.

2.3.4. Web Engineering as a Solution
With the popularity of the Web reaching new heights, the need for systematic, measurable and repeatable development processes is apparent (Murugesan & Ginige, Web Engineering, 2001). Web developers need to follow best practices, use better development and planning tools and be disciplined in their approach to creating Web systems.

According to Murugesan, Web engineering provides for these needs with a focus on long-term sustainability in developing Web-based systems and applications.

“Web Engineering uses scientific, engineering, and management principles and systematic approaches to successfully develop, deploy, and maintain high-quality Web systems and applications.” - Murugesan et al., 1999

The objective of WE, he says, is to minimize risks, improve quality, maintainability, scalability, all of which lead to a more controlled environment for Web-based application development. The core of Web engineering is to successfully manage the diversity and complexity of Web system development
life cycles, with the goal of identifying and recognizing potential failures before they manifest.

### 2.3.5. An Agile Approach

A report by Andrew McDonald and Ray Welland titled *The Agile Web Engineering (AWE) Process*, distinguishes between Web development and traditional software development. Although the two share certain characteristics, the report describes Web projects as typically having:

- Short development life cycles times (±3 months)
- Systems that integrate software and data
- Multidisciplinary teams

It further states that in order for Web-based applications to be more successful, Web-based projects need to focus on:

- A more thorough requirements analysis
- Better testing and evaluation of Web-based deliverables
- Issues associated with the evolution of Web-based technologies

McDonald and Welland justify an Agile approach to Web development using the above success criteria.
2.4. Agile Web Engineering

2.4.1. Overview of AWE

Agile Web Engineering (AWE) is a methodology based around The Manifesto for Agile Software Development. The purpose of the manifesto is to promote:

"... better ways of developing software by doing it and helping others do it. Through this work [developers] have come to value:

(i) Individuals and interactions over processes and tools
(ii) Working software over comprehensive documentation
(iii) Customer collaboration over contract negotiation
(iv) Responding to change over following a plan

That is, while [they] value the items on the right, [they] value the items on the left more."

McDonald and Welland are aware that the developers and the organization play the most important role in the success of a project, considering the methodology used to carry out the project as having a secondary impact. For this reason, they believe that the Agile route lends itself to (i). Furthermore, they highlight the shortfalls of Monumental processes over Agile, indicating that developers are misguided by the purpose of documentation, using the processes not as they were intended by the designers. Points (ii) to (iv) plus 12 other principles are justified through a high level explanation of what AWE tries to achieve.

They go on to discuss in great detail each phase of the AWE methodology. An overview of this, the Web Semantic Design Methodology (WSDM) and a User Centred Design (UCD) Approach to Web-based Software Development make up the remainder of this literature review.
2.4.2. Components of AWE

The AWE methodology is separated into seven phases, with one main deliverable - the Web application itself. Although supporting documentation may be beneficial, it is not required.

![Figure 1: The AWE Development Lifecycle](image)

- **Business Analysis**
  The objective of the business analysis is to extract a set of problems that must be addressed by the Web application.

  The Web application needs to meet clearly defined business objectives for it to have a metric for success. These objectives should be based around gaining a competitive edge in business, or some other equally important advantage.

  The greatest problems should be tackled first, allowing for potential changes to be made early on.

- **Requirements Analysis**
  The requirements phase is two-tiered. Firstly, it indicates what the proposed solution will do, which is termed functional requirements.

  Secondly, it describes what constraints are imposed on the solution, which are known as non-functional requirements. These include addressing performance issues, usability concerns, and security vulnerabilities.

  Thereafter, the team should start planning tests to determine criteria of what the right product should do.
iii. **Design**
A high level design, independent of lower level implementation details, is the core of this phase. Major issues concerned with building a complex Web application are addressed here. These include scalability, portability and redesign problems.

It is important to understand that for the above issues to be addressed, great attention must be given to this phase.

iv. **Implementation**
An alternate definition describes implementation as design at a low level. AWE suggests paired programming to build the Web application, as it provides extensive testing of features from the perspective of different developers.

v. **Testing**
A functional assessment of the features is vital in determining whether or not what has been built thus far satisfies the project's requirements.

Non-functional testing should assess at a minimum
- Application performance
- Cross-browser compatibility
- Scalability for the expected target audience
- Application security

vi. **Evaluate**
The evaluation plan is guided by the business and requirements analysis. It considers end-user usability, an evaluation of which often leads to a greater understanding of the problem space. Helpful feedback is put back into the analysis phases, and the analyze-design-implement-test-evaluate cycle repeats.

Evaluation has proven to be a time consuming, expensive task, and thus is not required for every iteration of AWE. This, however, has the implication that the success of the project is at risk, as there is a greater chance of over- or under-engineering the product, or the product not meeting the users' needs.

vii. **Deploy**
Initially, the Web application is deployed as a trial in real world scenarios. The overall performance is evaluated, and shortfalls of the system are recorded, improved upon and after the system is revised, updates are rolled out.

2.4.3. **Summary**
AWE provides a detailed, systematic set of steps to support the successful development, deployment and maintenance of Web applications as per the end-
users request. While it is quite laborious, the objective of such a thorough methodology is meant to identify and correct any misunderstandings, risks and hidden problems. Each iteration should focus on the problems that pose the highest risk to ensure that efforts on previous iterations are not wasted.
2.5. Web Semantic Development Method

2.5.1. Overview of WSDM
WSDM, formerly called the Web Site Development Method, offers a systematic, multi-phase approach to Web design. In a paper titled *Semantic Web Development using WSDM* (Plessers et al, 2006) the authors present a methodology comprising five distinct phases, each focusing on a particular aspect of the Web design cycle.

2.5.2. Components of WSDM

- **Mission Statement**
The mission statement describes the subject of the website, the purpose it must fulfill and it identifies the target users.

- **Audience Modeling**
The targeted users are classified into audience classes, which is described as a group of visitors that have the same knowledge base and functional requirements.

Audience classes can be further classified into subclasses, where the audience is more specific in terms of their requirements. The characteristics and usability requirements are also expressed here.

The model that represents the audience hierarchy, which includes their characteristics and requirements, is called the audience model.
iii. **Conceptual Design**
A high level abstraction of the content, functionality and structure of the Web system is defined. Conceptual design is divided into Task Modeling and Navigational Design.

The Task Modeling phase is where the content and functionality are defined. The purpose of this phase is to determine the different tasks that different audience classes must perform, considering the requirements formulated during audience modeling. Each task is broken into elementary subtasks. For each subtask, an object chunk is created that formally describes the functionality required of the subtask.

The goal of Navigational Design is to define a conceptual structure of the Web system, thereby modeling how the audience navigates through the system and perform their tasks. Navigation structures are defined for each audience, indicating dedicated navigation structures.

iv. **Implementation Design**
Conceptual design models are completed with the relevant information required for the actual implementation. The implementation design comprises three sub phases, namely the System Structure Design, Presentation Design and the Data Source Mapping.

The above processes map the conceptual model of the system onto Web pages, taking into consideration design patterns, and the presentation of different information for different audiences, devices, contexts and platforms.

v. **Actual Implementation**
The Web system is implemented according to the implementation design, which serves as a detailed project skeleton.

2.5.3. **Summary**
WSDM takes a new approach to Web development, relying on various models to determine the best way to represent the data. It allows a web developer to describe the Web application from different perspectives and at different levels of abstraction. Furthermore, it provides a linear, systematic way to develop Web applications.
2.6. A User Centered Design Approach

2.6.1. Overview of UCD
UCD is an iterative, audience driven methodology with a large focus on HCI\(^1\). In an article from the Journal of Software Engineering and Applications, *Exploiting User Centred Design Approach and Interactivity in Web-based Software Development*, the author centres his approach around *Uses and Gratification Theory*, which is an accepted theory to understanding mass communication. This theory aims to explain how people use media for their needs and gratification, focusing on the user instead of the actual message (Siricharoen, 2011). He classifies the needs of people into five categories; cognitive, affective, personal interrogative, social, and tension-free needs, each need a different motivation for visitors.

There are other HCI components that the author talks about, but these are out of the scope of this literature review.

2.6.2. Components of UCD
As per the journal article by Sirichareon, UCD is broken down into four practical phases.

\(i\). Analysis
Analysis activities include user and task analysis, as well as analysis of the business requirements.

Research takes the form of questionnaires, interviews, focus groups and other requirement gathering techniques. These techniques are used throughout the analysis-design-impliment iterations to gather information, allowing developers to effectively meet the needs of users.

\(ii\). Design
This phase is one of the most iterative as it starts off with a simple prototype that evolves until all the critical design decisions have been made.

\(iii\). Implementation
The system is implemented and revised in cycles of analysis-design-impliment. Each iteration is a step closer toward better matching the user specification.

\(iv\). Deployment
The Web application is deployed, evaluated and revised in contribution to the long-term sustainability of the product.

---

\(^1\) HCI (human-computer interaction) is the study of how people interact with computers and to what extent computers are or are not developed for successful interaction with human beings (Rouse, 2005)
2.6.3. Summary
User Centric Design processes concentrate on the audience throughout the development lifecycle. The core of UCD involves first and foremost considering the prospective audience and considering the goals of the audience when they visit the Web page. Identifying the information that visitors will likely need is important, as this information must be made accessible and understandable.
2.7. **Comparison of Methodologies**

Three distinct Web development methodologies have been discussed separately in this paper. This section aims to compare the different methodologies and determine which option is best for developing complex Web applications with different interacting technologies, meant for two types of users. The two types of users are members and visitors.

While the AWE methodology is the most detailed, the trade-off is the relative additional time and effort invested in requirements elicitation and revision. In terms of complex Web applications, the greater the detail, the better. Coupled with close collaboration with the client, it is hard to ignore this as the most methodical approach to a successful Web system.

UCD on the other hand is a flexible approach that puts the user first. A design primarily based around the users needs is intended to provide the best kind of user experience. However, due to the complex relationship between humans and computers, as well as the general nature of humans, this could result in constantly changing requirements.

Both processes are iterative, with UCD following a similar structure to AWE, with a smaller emphasis on testing and evaluation.

WSDM tries to tackle Web development from a different angle, with a relatively high amount of documentation as compared with the other methods. There is a large amount of estimation, as the method attempts to predict what kinds of functions are needed based on an analysis of the targeted audience. It also tries to achieve additional semantic annotation, which is out of the scope of this literature review, but must be mentioned for completeness.

The linear nature of WSDM means it falls short in terms of customer feedback. Much of the effort when developing using this method goes into modeling relationships between objects. This could be a useful tool to integrate different technologies such as a database with a Web development environment. However, the linearity is not favourable for Web development.

A combination of the AWE and UCD should result in the best methodology for Web development of complex applications. The idea is that UCD serves as the basis for which AWE can revolve, as UCD does not consider a host of factors required for successful deployment. This includes factors such as scalability, security, performance, and maintenance.
2.8. Conclusion

As the Web pervades us, our dependence on it increases. The demand for Web applications has multiplied exponentially in the last two decades, attracting Web development from various professions.

There is a legitimate and growing concern about the ad hoc manner in which Web-based applications are currently being created. Greater sophistication coupled with an increasing complexity of new Web-based applications has resulted in many new challenges that need to be satisfactorily managed or addressed.

Web-based system development is not just graphic design or content development any more. There are a growing number of complex applications, the success of which is dependent primarily on the development team, and secondarily, on how the project is managed.

Evidence suggests that an Agile methodology to Web development is best, as it allows for rapid prototyping, extensive testing and incremental improvements.

Three methodologies were then presented, two following an Agile methodology and the other, a traditional methodology. AWE turned out to be the most detailed, resulting in the best methodology to handle complexity. WSDM and UCD focused mainly on the end-user, which had the merit of producing a system accurately in line with the customer specification. However, the linear nature of WSDM did not leave room for much improvement, and thus UCD was preferred over WSDM.

Overall, the UCD methodology formed the core of the development methodology when required to make a complex Web application. The AWE process was followed to supplement important factors not considered in UCD, such as testing and evaluation.
3. Requirements Analysis

3.1. Business Case Summary
The business case for the proposed system was introduced in the first chapter. For ease of reference, the most important expectations of the Administrator subsystem have been summarized below:

Derivco need a web-based Administrator system with tools to setup and manage players, their profiles, and tournaments, results and fixtures.

The tools should
1. Allow users to register and manage their account
2. Allow tournaments to be created and managed
3. Be able to track a tournaments progression
4. Consider when players are available as well as account for differing time zones when scheduling fixtures (so that players don’t end up playing at inappropriate times)
5. Have a mechanism in place to log and handle match result disputes

3.2. Requirements Elicitation
The core requirements for the project were determined through the project description. This document briefly described the nature of the project and gave insight on the size of the project as well as the basic features expected. These requirements were further elaborated on through Skype interviews with a representative from Derivco, Douglas Wilson. Douglas was our primary source of requirements elicitation and provided information pertaining to what technologies we should use and other non-functional requirements. He was also open to email communication for any clarifications regarding functional requirements.

The first Skype interview with Douglas took place a few days after we were given the project description. In this interview, he introduced Derivco, explaining who they were, what they did and what the purpose of the project was from Derivco’s perspective. He also clearly defined what was in the scope of the project and what was not. As we had already read the project description, we had an idea of what the project was about and thus we were able to use this session to clarify any first impression misunderstandings that we had.

The next Skype interview was conducted two months later, after we had a bit of experience familiarizing ourselves with the Microsoft .NET framework. We had more technical expertise under our belts, and could thus ask more technical questions about the implementation of the features that were required. The
The purpose of this interview was simply a progress check, to explain to Douglas what we had done so far, and what we would be doing in the next few weeks.

In between Skype interviews, we contacted Douglas as we needed him, via e-mail. These were generally brief conversations about specific functionalities that needed clarification, ranging from issues such as dispute resolution workflows to security authentication troubleshooting.

To gain a better understanding of what the common features of an eSports management portal was, we looked online for existing systems. We came across two in particular that exemplified what we would like our system to be like, namely toornament.com and binarybeast.com.

Toornament gave access to a multitude of tournament types, with intuitive drag and drop interfaces and customizable tournament formats. However, it did not have support for fixture scheduling, either locally or on a global scale (Toornament, 2014).

BinaryBeast focused on having a few specific tournament types, but excelled in displaying tournament progression and leaderboards. However, it too, did not have support for fixture scheduling (binarybeast, 2014).

From looking through systems that offer similar functionality to ours, we were able to build an intuitive look-and-feel, first using paper prototypes, and then using CSS and HTML. Connor Patrick handled most of the interface development, and more about this can be found in his report.

### 3.3. Core Concepts and Terminology

Early on in the requirements analysis it became necessary to formulate a glossary of terms to clearly distinguish many seemingly similar terms. This follows below:

- **Administrator**: A super-user of the system that has access to privileged functions.
- **Availability Window**: A set of times over the week when a user prefers to play.
- **Dispute**: When players upload match results that are not a true reflection of what actually happened.
- **Fixture**: A date and time that match is scheduled to take place, similar to a match.
- **Game**: A particular game title that is played.
- **Knockout**: A tournament that excludes the users from the next round if they lose.
- **League**: A representation of a game.
- **Match**: A description of who is playing, on what date and at what time.
- **Player**: A normal user of the system.
**Scheduling**: Refers to setting up a fixture list based on a player’s availability window.

**Tournament**: A set of matches within a league that a group of teams participate in.

### 3.4. Functional Requirements

The following is a list of the most important requirements that would be needed by the administrator, the player and the mobile component respectively.

#### 3.4.1. Administrator Requirements

1. **User management**

   The administrator must be able to edit a user’s details and remove users from the system.

2. **Tournament management**

   Tournament creation and management of existing tournaments needs to be handled by the administrator. This includes specifying tournament names, choosing a start date and an end date and choosing what type of tournament is to be played.

3. **Scheduling fixtures**

   Fixtures need to be scheduled according to when a player is free, a set of times known as his availability window. The scheduling system must retrieve his availability window, account for time zone differences, compare it with his competitor and schedule a match when there is an overlap between windows. It must also schedule matches evenly between the specified tournament start date and end date. Furthermore, matches should not take place on a Sunday.

4. **Dispute resolution**

   Should there be a discrepancy between the results uploaded by the players, there needs to be a system to acknowledge and resolve disputes.

#### 3.4.2. Player Requirements

Player functionality must include the ability to

- Manage his profile and password,
- Join or leave leagues,
- Create, join or leave teams and tournaments,
- View tournament leader logs,
- Upload proof of results,
- Log a dispute if uploaded results clash.

#### 3.4.3. Mobile Requirements

Mobile functionality must include the ability to

- Take pictures of the match result and upload it to the Website,
- View the fixture schedule for upcoming matches and tournaments,
- Log disputes

3.5. **Non-functional Requirements**

Where functional requirements describe what the system should do, non-functional requirements elaborate on how the system should do it. The following is a list of the design considerations that the project is based on.

3.5.1. **Maintainability**

Maintainability relates to the degree of complexity required to understand the logic behind lines of code, isolate and correct defective code and implement future requirements on top of the existing functions. Derivco specialize in the Microsoft ASP .NET technology stack. Thus for the purpose of easier maintenance at Derivco, Douglas specified that the system should be created using Microsoft Visual Studio 2013, which hosts the Microsoft .NET Framework.

3.5.2. **Data**

The project requires data about users, matches, results and disputes to be stored in a relational database management system (RDBMS). The RDBMS will be hosted in-house on the Derivco servers.

3.5.3. **Usability**

Usability of the system is dependent on how easy it is to navigate, understand and use features. It also depends on the aptitude of the people using the system in using Web-based technology. Since the clients are software developers themselves, we can assume that they are familiar with Web applications. We can thus implement additional dimensions of complexity on top of the core system requirements to improve the user experience.

3.5.4. **Scalability**

Scalability in this context refers to the ability of the system and its software to sufficiently handle a growing number of users and tournaments. The system will need to support upward of 1000 users according to Douglas. In addition, the servers will be distributed across the world, as this parallels to the many different locations where staff are based.

3.5.5. **Performance**

Performance refers to how fast Web pages display as well as the speed at which business logic is performed.

3.5.6. **Security**

It is required that each user has a unique profile based on his particular details. These details are somewhat sensitive and should not be accessible by other users. Thus, a password based login system is required for every user.
Furthermore, there needs to be a distinction between two roles, the administrator and the player. The administrator has access to all the functionality of the player in addition to privileged functionality that the player cannot access. It is imperative that the player is not able to access these functions.

3.6. Use Cases
All of the use-cases for the administrative role of the eSports Portal are shown in Figure 3 below, and the most important use-cases, highlighted in red, are described in detail thereafter.

Figure 3. Administrator Use Cases
3.6.1. Create a User
Actor - Administrator
Description - A user wants to create another administrator
Flow - Administrator Tools → Manage Accounts → Create User → Enter Details → Submit

3.6.2. Create Tournament
Actor - Administrator
Description - A user wants to create a tournament within a league
Flow - Administrator Tools → Manage Tournaments → Create Tournament → Choose League → Enter Tournament Details → Submit

3.6.3. Schedule Knockout Tournament
Actor - Administrator
Description - A user wants to generate a fixture list for one round of a knockout tournament
Flow - Administrator Tools → Manage Tournaments → Schedule Round

3.6.4. Schedule Round-robin Tournament
Actor - Administrator
Description - A user wants to generate a fixture list for an entire round-robin tournament
Flow - Administrator Tools → Manage Tournaments → Schedule Season

3.6.5. Resolve Disputes
Actor - Administrator
Description - A user wants to settle a dispute between two conflicting results
Flow - Administrator Tools → Manage Disputes → View Dispute Details → Contact Teams Captains → Resolve Dispute
4. eSports Portal Planning and Design

4.1. General Overview

After extracting the requirements and converting them into system specifications, we began the design phase. This chapter covers the development methodology used throughout the software development lifecycle and how product backlogs were constructed and prioritized, specifically in the administrator domain. The version control tool is then briefly described, after which the actual planning and design is explained in depth.

4.2. Agile Development Methodology

Due to the project length, a flexible development methodology was required. Thus, we used an agile methodology, particularly with a SCRUM-esque approach to consolidate code and knowledge. This was in addition to weekly meetings with our supervisor, Sonia Berman, where we would discuss which feature to develop next.

Nine SCRUM sprints of two week time-boxes each were planned, three sprint cycles at a time. This required us to create a prioritized product backlog, which was broken down into a sprint backlog. Sprints one, two and three were planned first. At the end of sprint three, sprint four, five and six were planned. At the end of sprint six, the final three sprints were drafted.

As sprints were completed, functionality that showed growing importance or complexity were brought forward, tasks that could be pushed back were deferred and others that were irrelevant were excluded completely. A screenshot of the first sprint cycle is exhibited in Figure 4, and the remaining two sprint cycles are documented in the appendix <Appendix A>.

Figure 4. Sprints One, Two and Three
An online, agile management application called Agilefant was used to track sprint cycles and backlogs. It was the most comprehensive and user-friendly tool that we found, after looking through ScrumDo.com and SonicAgile.com. It was also free for five users, which meant we could host all of the component backlogs in a central place.

BitBucket, in combination with the desktop application SourceTree, was used as version control to host the project repository, as my team members had some experience using Bitbucket before.

### 4.2.1. Agilefant

Using Agilefant, we would record our progress by creating tasks within stories that correspond to functions within features. Tasks and stories were assigned points depending on their relative size and complexity. As functions and features were completed, they were marked off as done. Agilefant would then graph our work spent on features and indicate how much still had to be done within the sprint cycle. This was represented in a burn-down chart, to easily show whether we were working hard enough to meet project deadlines.

Agilefant proved to be an effective way to manage our time and helped keep the project on schedule. It was also great as a collaborative tool as it showed incrementally how the team was bringing the project together.

### 4.2.2. Version Control

BitBucket provided the team with a way to integrate the individual administrator and player components into a single project remotely. At first, we had trouble using the tool, but as we learnt more about how to use its functions, we were effectively able to track changes that were made on each part, and merge these as if we were working on the same computer.

We started off with a basic authentication interface as the master because this functionality provided a base to both the administrator and player components. From there I created a branch for the administrator interface and Connor did the same for the player interface. As more features were developed, code would be committed with a brief description of what had been added, and this code would be pushed to our respective branches. This kept us up to date with the latest changes done to the project and ensured any new code was not overwritten.

At the end of the initial sprint cycles, code was manually merged with the master code. However, we were able to use more of the functionality provided by BitBucket as we learnt more about it, that allowed us to pull changes from other
branches, merge them into the branch we were using, and then merge the up-to-date version back to the *master*, remotely.

Code that was written for the mobile application was not included in BitBucket. It would be periodically merged manually. This did not pose a problem as the mobile code was separate from the administrator/player code, and thus we could integrate the code without worrying about breaking the entire system or unintentionally overwriting new administrator/player code.

**4.3. Privilege Control**

Access to administrator-privileged functions was controlled using roles. Users were either assigned a player or an administrator role. Administrators had access to an “Administrative Tools” tab, from which they were able to manage the system database, schedule tournaments and resolve disputes. If a player was able to identify the URL that routed to an administration related function, they would be redirected to a “No Access” page.

**4.4. Design Phase**

After familiarizing ourselves with the Microsoft .NET framework using the *Introduction to MVC 5* tutorials, we discussed how we were going to apply the MVC design architecture to our project. A typical MVC application is characterized in Figure 5 (CodeProject, 2012).

**4.4.1. Model-View-Controller (MVC) Architecture**

The MVC architecture was adopted because it provides a clean separation of presentation code, business logic and data access functions. Organizing the code in this way increases its readability and understandability making it easier to maintain, which is a non-functional requirement specified by Derivco. Each layer is described in detail below.

**i. Models**

Models make up the data access layer as seen in Figure 5. The .NET framework uses a Code First Entity Framework, which means that models of domain objects are coded first before they are created on the database server. Domain objects are programming objects modeled against

---

![Figure 5. MVC-layered Architecture](image-url)
real world objects, and contain the attributes representing real world information, as well as the getters and setters (accessors and mutators) for each attribute. The getters and setters allow data within objects to be accessed and passed to the controller.

Microsoft Visual Studio scaffolds a database table for each model, creating columns for each particular attribute. Razr code within Visual Studio also allows for data validation, which throws a descriptive error message to the user if the wrong type of data is entered.

**ii. Controllers**
The controller is responsible for creating, reading, updating and deleting (CRUD) data in the database. The business logic behind every feature is written here, and thus it makes up the business layer. As its name specifies, it controls what happens to the data after it has been accessed from the database and before it is sent back.

**iii. Views**
The presentation layer is what the user sees and interacts with. These are made up of files that contain C#, HTML, CSS and Razr code and are used to control how Web pages are presented. Style sheets are used to present a consistent system-wide look and feel, while a combination of HTML and C# code is used to further control how data is presented. Razr code is used for input validation.

**4.4.2. Conceptual Models**
The first design step involved converting the requirements derived from the requirements analysis into separate concepts that could then be used for the entity model, and ultimately the entity-relation model.

**i. User Accounts**
- Are created by administrators or players
- Have a user profile
- Have a default, customizable availability window
- Are either a player or administrator
- One user of the system is assigned to one user account

**ii. Leagues**
- Are created and managed by administrators
- Represent a game title
- One game title is assigned to a league
- Many users can join one league

**iii. Teams**
- Are created and managed by players
- Can be joined by players
- Have a team captain, who is the person that created the group
- Many users within a league can join one team

**iv. Tournaments**
- Are created and managed by administrators
- Can be joined by players
- Have two types, knockout and round robin
- Are scheduled by administrators
- Many teams can join one tournament
- Many tournaments can be created within one league

### 4.4.3. Entity Models

An entity model describes the structure of entities in the system, encapsulating the characteristics and functionality of real world objects. It is a representation of the business domain in terms of the proposed system.

The purpose of the entity model is to represent how entities can interact with each other to realize use cases. The result is an entity-relational model, which is used to implement the database schema.

The entity-relation model is depicted in Figure 7 and is described on the page that follows.
Figure 7. Entity-relation Diagram
i. **Users**
Users have a unique username, a contact email address, a field to specify what time zone they are in and a default availability window. This forms their user account.

ii. **Leagues**
A league is created by the administrator and represents a certain game. This is stored in a League table. Players join leagues if they want to participate in any tournaments that are created. Users that are part of a league are recorded in a LeagueMembership table, which links usernames with a unique league identifier called LeagueID.

iii. **Teams**
Once users are part of a league, they can create and form teams with other users in that league. A Team table shows which teams are in a league and a TeamMembership table links users to teams. Teams are identified with a unique ID called TeamID.

iv. **Tournaments**
Tournaments are created within leagues and are identified by a unique TournamentID. A list of tournaments is stored in a Tournament table. These records specify the league that the tournament is a part of, the number of teams that the tournament can hold, the tournament type and the start and end date of the tournament.

Teams join tournaments and this relationship is recorded in a TournamentMembership table. It forms a relationship between TeamIDs and TournamentIDs.

v. **Match Fixtures**
Fixtures are scheduled depending on when players are free and according to what type of tournament is being played i.e. either knockout or round robin. For functional purposes, when teams that have more than one user play, scheduling considers only when the team captain is free. Fixtures are scheduled uniformly between the start and end date, and not on Sundays, with no more than one match per tournament taking place on a particular day. Match fixtures are stored in a Match table.

vi. **Match Results**
The results of matches are stored in a MatchResult table. It identifies the winner and the loser as well as the score, making it easy to eliminate teams for knockout or record points for round robin. Uploading screen shots of the final score serves as proof. This is stored as a file path to the folder on the server. Should there be a discrepancy between uploaded results, a dispute is logged.
vii. Disputes

A Dispute table records disputes as they are logged, storing the date and time that the dispute was logged, the MatchID of the match in question and the dispute status. The status can either be “dispute logged”, which is performed by a player or “resolved”, which is done by an administrator.

4.4.4. Site-Map

Figure 8. Site-Map of Administrator
5. Implementation Phase
Having planned our individual sprints and the basic system structure that would meet the core requirements of Derivco’s needs, we began to implement the designs. Chapter 5 discusses the development phase chronologically, describing each functional implementation as sprints happened. Only the sprints relevant to the administrator functionality are included in this report.

5.1. General Overview
The first three sprints (sprint cycle one) built up to the first functional prototype, which was a simple Web application that allowed administrators to login and create and manage users and leagues. The next three sprints (sprint cycle two) were reserved for building on this functionality, adding administrative tools to create and manage tournaments within leagues. Different scheduling and dispute algorithms were designed in these sprints as well. In the final three sprints (sprint cycle three), scheduling and dispute algorithms were selected, implemented and tested.

Since the MVC design architecture was used for this project, I will discuss each sprint according to how models, controllers and views were developed on a feature-by-feature level.

5.2. Sprint Cycle One
Creating create, read, update and delete (CRUD) functionality for the administrator was the focus of the initial two sprints, with the third sprint left for the implementation of the security framework.

5.2.1. Administrator Users and Account Management
   i. Administrator Model
The model for an administrator is identical to a players model, the distinguishing field being the role to which a user is assigned. As explained in 4.4. above, administrators have privileged access to certain functions. It contains basic information about a user, including his First Name, Last Name, Email Address and the Time Zone that he is in.

Each user is associated with an availability window as well, which contains the username of the user along with six days, Monday to Saturday, recording when a player is free.

Representing when a player can compete was tricky. Initially, the algorithm was designed to store the times that a player can play at in a string with the digits of the hour that the player is free, representing availability. For instance, if a player was free on Monday at 09:00, 12:00 and 14:00, then his availability would be stored as 091214 under the Monday column. This proved hard to work with, as it
was difficult to make comparisons between players availability because the strings were too irregular.

A better option was to store each day as a 12 digit binary string. Each value represented whether a player can compete during this hour, with 0 specifying that he cannot and 1 specifying he can. The first digit represents 8:00a.m. local time, with subsequent digits corresponding to subsequent hours. For instance, if a player was free at 09:00, 12:00 and 14:00, his availability window would be recorded as 0100 1010 0000. Using this representation makes it easier to compare availability windows among users, because every user has a uniform length availability window. This is used later to schedule matches.

ii. Account Controller
The administrator can setup new user accounts and manage existing user accounts through the Account controller. Creating and managing users involves editing their user details, and deleting user accounts. An administrator can also either make a player an administrator, or make an administrator a player.

5.2.2. Leagues
An administrator first and foremost needs to be able to create and manage leagues (synonymous to a specific game), which was modeled through the League Model and facilitated through the League Controller.

i. League Model
Leagues are identified by a unique LeagueID and contain information including the League Name and the Start Date of the league.

ii. League Controller
The league controller has four main methods, one for each of the CRUD functionalities. An administrator is able to view all of the leagues in a list, view specific details about a particular league and create, edit and delete leagues.

5.2.3. User Authentication
There were two possible choices of user authentication offered by Microsoft Visual Studio, authentication using forms and using Windows authentication.

i. Forms
Forms authentication is based on session cookies. Credentials such as the username and password are collected directly from an HTML form on the client side and sent to the business logic of the application for authentication. If authentication is successful, the application issues the client a cookie that is used for subsequent resource requests. If the cookie expires or the request for resources does not contain the cookie, the application returns a redirect to either an error page or a login page. The application stores valid credentials either in a configuration file or a SQL Server database (Microsoft, 2003).
Although Derivco requested that the system make use of Windows Authentication, the Android application did not support it. The primary requirement for Windows Authentication is that users must have a Windows account and this is not possible on an Android based phone. Thus, for the sake of the project, authentication using forms was chosen.

The advantages of using forms for authentication are personalization i.e. users can choose their usernames, and a Windows account is not required. The disadvantage is it poses a security risk, especially for replay attacks, for the lifetime of the cookie, unless using SSL/TLS.

To implement forms authentication a logon page must be created and unauthenticated clients must be redirected here. You must also create your own scheme for account authentication.

**ii. Windows Authentication**

Windows Authentication uses Internet Information Services (IIS) to authenticate the client. IIS is a Web server created by Microsoft that supports ASP. It was the preferred authentication for the system as IIS provides tight integration with the ASP.NET framework and Windows, making it an ideal authentication service for Derivco. However, it did not support authentication from Android devices.

It works by passing a security token to ASP.NET after IIS authenticates the client. ASP.NET then constructs and attaches an object of the WindowsPrincipal Class to the application context based on the security token it receives from IIS. Through this process the user is allowed to access the required resources (Microsoft, 2003).

The advantage of this approach is thus that it uses registered Windows accounts for authentication, eliminating the need to write custom authentication code. Disadvantages are that a user has to have an individual Windows user account and cross compatibility with Android devices is not supported.

### 5.3. Sprint Cycle Two

Functionality to manage users and leagues was the base on top of which the rest of the system was built. In sprint cycle two, the focus was on creating tournaments within leagues and designing scheduling algorithms based on when the tournament takes place, who was in these tournaments and when they were available to compete.

---

2 The WindowsPrincipal Class enables code to check the Windows group membership of the Windows user that is currently logged on to the machine.
5.3.1. Tournaments and Scheduling

i. Tournament Model
The purpose of the tournament model was to have a structure that could link with users profiles and hold all of the relevant details needed to create a tournament schedule. Tournaments are identified by a unique TournamentID and Tournament Name. Other details included:

- The League Name that the tournament belongs to,
- The Start and End Date of the tournament, used to schedule matches evenly for the tournament duration
- The Type of Tournament, either knockout or round-robin, is used to determine how matches would be scheduled,
- The Capacity of Teams the Tournament Allows, used to determine the number of places left in a tournament i.e. how many teams can still join that tournament (maximum tournament size),
- The Current Number of Teams that have joined the tournament,
- The Number of Users in each Team Required to Join a Tournament (minimum team size) and finally,
- The Current Round Number, used to track the tournaments progression

ii. Tournament Controller
The tournament controller initially facilitated the usual CRUD operations. Later in the development lifecycle, the scheduling functionality was implemented here as well.

iii. Scheduling
At the end of the second sprint cycle, two scheduling algorithms were designed, each with varying levels of complexity. These algorithms are explained below.

Algorithm One
The system schedules the matches to be played as a fixture list, which specifies the time as well the day that the match is played on. Matches can happen at any time between 08:00 and 19:00 on any weekday except Sunday. Note that all members of each team have to be available at the same time for a match to be scheduled successfully.

The time is determined by trying to find an overlap between pre-specified availability windows that each user in every office (across the world) must indicate. For instance, someone in Hong Kong (GMT+8) would specify that they can only play between 10:30-13:30 and 15:00-17:00 (both GMT+8) and someone in Durban (GMT+2) would specify that they can only play between 12:00-14:00 (GMT+2), the system will schedule the match by calculating the overlap.
In the #Overlap example, there is an overlap between the windows 07:00 to 09:00 & 08:00 to 10:00, which means the match will be held at 08:00 [GMT].

Thus, the user in HKN will see the match as scheduled at 08:00 + 8 = 16:00 in HKN and the user in DBN will see the match as scheduled at 08:00 + 2 = 10:00 in DBN.

This option seems to complicate the scheduling system because it requires availability to be stored for individual days. Furthermore, to solve the No Overlap cases, requiring the administrator to reschedule the match adds another layer of complexity.

**Algorithm Two**

The system schedules the matches to be played as a fixture list, only specifying the day that the match must take place. It is up to the players to decide when it happens, as long as it happens on that day. This means that players have to coordinate among themselves when to play matches and get back to the administrator for every match, which requires a lot of the administrator's input.

**Evaluation**

Algorithm Two was seen as undesirable for Derivco because they want the administrator to be involved as little as possible in scheduling the matches. While Algorithm One involves more complexity, it was chosen as the algorithm best suited for Derivco needs and thus best suited for the project.

In both algorithms, the system will schedule matches evenly between the start date and end date of the tournament. It might seem intuitive that matches should be scheduled as soon as possible, however, it was a requirement by Derivco that
matches be evenly spaced between the start and end date, with no more than one match per tournament per day. The reason for this is that it leaves time for matches to be re-scheduled, should one of the teams not be available to compete.

It must be emphasized that opponents are not chosen based on when they are available. If this were the case, players would repeatedly play the same opponents, because it would search for players with similar free time slots. This is undesirable as I wanted as much variation between who plays who as possible.

5.4. **Sprint Cycle Three**

After choosing a scheduling algorithm, implementation of the algorithm began. This was the prime focus of sprint seven, or the first sprint of the third sprint cycle. The design and implementation of the dispute resolution workflow was completed in the following sprint, and both scheduling and dispute resolution were tested in the final sprint. The testing phase is discussed in the next chapter.

5.4.1. **Core Tournament Scheduling**

The primary issue in scheduling tournament matches was finding an appropriate time when both competitors are free, even though they are in different time zones. The next challenge was developing the algorithm in such a way that the hit rate of finding free time overlaps was high enough in order to minimize administrator regulation i.e. minimize the amount of rescheduling due to matches scheduled when players are not available. The third primary requirement considered is that matches need to be scheduled evenly between two dates, the specified start and end date. The scheduling algorithm was built around addressing these three core complications.

i. **Availability**

As mentioned in 5.2.1., every user account is associated with an availability window, which is an indication of the times in the week he prefers to play. This is recorded in six columns, which corresponds to each day of the week excluding Sunday. Each column contains a binary string representing times that a user prefers to play, with the first digit representing 8:00, local time. An example would be the binary string 0100 1011 0101 under the column Tuesday, which implies that they are available to play at 9:00, 12:00, 14:00, 15:00, 17:00 and 19:00.

These binary strings are read from the database and stored in a two dimensional Boolean array, the first dimension representing the day, and the second the times, with false representing that they do not want to play in this period, and true representing that they do. This array is passed as a parameter, along with the team captain’s username, the team’s name and the team’s time zone, to an object of the AvailabilityWindow class. The AvailabilityWindow class is described in ii. below.
**ii. AvailabilityWindow Class**

The AvailabilityWindow class has two primary fields, and is supported by two secondary fields. The first primary field is a Boolean 2D array, which stores a user’s availability and is compared against other users’ AvailabilityWindows. The second primary field is the time zone value that is added to GMT, that gives the user his local time. This value is used to convert his local time to GMT+0 so that users in different time zones can have their availability compared in a universal time context. The two secondary fields are used to keep track of whose availability is being compared.

There are two constructors. The first constructor takes in four parameters, the username, the team name, a time zone value and a Boolean array. This constructor is used in the tournament controller to store read-in availability values. The second constructor takes in one AvailabilityWindow object and creates an AvailabilityWindow object accounting for the users local time zone. It constructs an AvailabilityWindow object as shown in Figure 9 below.

```java
// AvailabilityWindow constructor that creates a user Availability based on UTC
public AvailabilityWindow(AvailabilityWindow aW)
{
    timeZone = aW.timeZone;
    teamname = aW.teamname;
    username = aW.username;
    A: timeSlots = new bool[6, 48];
    B: for (int j = 0; j < 6; j++){
        C: for (int i = 16; i < 28; i++){
            D: timeSlots[j, i - timeZone] = aW.timeSlots[j, i - 16];
        }
    }
}
```

Figure 9. AvailabilityWindow second constructor

This constructor is used specifically in the calculation to determine where there is an overlap in availability.

At line A, an oversized 2D Boolean array is created. The reason for its excessive size is so that no available times are lost in the conversion to GMT+0.

Line B is a loop that iterates through each day of the AvailabilityWindow object.

Line C is a loop that iterates through each period within a day. It starts at 16 to allow for time zones in GMT+16, and ends at 28 so that all twelve time periods are captured i.e. the code loops 12 times.
Line D assigns each value in the inputted AvailabilityWindow object to the corresponding GMT+0 time period, setting up a new AvailabilityWindow object in time zone GMT+0.

In the calculateOverlap method<Appendix B>, each users availability window is converted to GMT+0 and used in the overlap calculation. Overlaps are calculated by iterating through each users new GMT+0 availability window and, where both users have a value of true listed in their specific time period on a specific day, a new AvailabilityWindow object called CommonTimes records the value as true according to that particular day and time index. This sets up an AvailabilityWindow object containing all common times that matches can be scheduled at between these particular two teams, in a universal time context.

This common availability window is then passed to a trivial method called getOverlaps <Appendix C>, which returns the potential match days and times as a String array. These were matched against a list of potential match dates (calculated separately as shown in 5.4.2. and 5.4.3.), and scheduled according to when the players are free depending on what day the potential match date is on. If no overlap on that particular date was found, getOverlaps returns -1 -1 and the match is scheduled for 13:00 GMT+0.

**iii. Minimizing No-Overlap Outcomes**

The nature of this availability problem is that if too few time slots in the availability window are marked as “able to compete”, the chances of finding a common availability window are diminish greatly. Using teams of two, random availability windows with at most four slots marked as “able to compete”, resulted in 20% of common times found. However, when seven or more slots were used, the rate of successful common times found increased to 80%. Thus, since Derivco is competing on a global scale, it is imperative that players choose an availability window large enough in order for fixtures to be successfully scheduled. In the event that a common availability window cannot be found, only the team captain’s availability of each team is considered. If this is still unsuccessful, then the match is scheduled automatically for 13:00, GMT+0.
5.4.2. Knockout Tournament Scheduling

A knockout tournament is a tournament where teams compete against each other, and only the winning team from each match moves on to the next round. This differs from a round-robin tournament, where every team plays every other team once, as explained in 5.4.3.

The next two sections describe the characteristics unique to the two different types of tournaments, knockout and round-robin. They differ in three fundamental aspects; the way they are scheduled, the way match dates are found and the way competitors are matched against each other.

For knockout tournaments, there are two regulations that must be adhered to for valid fixtures to be scheduled. The first is that the number of teams in the tournament must be a power of two. The second is that fixtures for the next round can only be scheduled once a round is over.

i. Scheduling Rounds and Match Dates

Knockout tournaments are scheduled as rounds because the result of each round determines who plays in the following round. Two multipliers were created, multiplier A that uses the round number to determine the length of each round, and multiplier B, that uses the round number to determine from when to schedule a round. Before these are explained further, it is important to identify some useful helper values.

Number of Matches = (n – 1).
The number of matches in a knockout tournament is dependent on the number of teams playing. If there are n teams playing, then there are a total of n-1 matches played altogether. For instance, if there are 8 teams playing, namely team A, B, C, D, E, F, G and H, the first round will have 4 matches, the next round will have half as many and the third round will be the final, which calculates to 7 matches.

Number of Rounds = (log₂(n – 1)).
The number of rounds in a knockout tournament can also be calculated given the number of teams. If there are n teams playing, there are a total of log₂(n) rounds. As seen in the example above, if there are 8 teams playing, there are 3 rounds that must be scheduled.

Length of Tournament = ((startDate – endDate).TotalDays).
The total number of days that the entire tournament was scheduled over was simply the difference of days between the start date and end date. However, the days over which each round takes place must also be calculated, as this is
different from the days over which the entire tournament is held. Calculating this was the primary use for multiplier A.

The definition of multipliers A and B are shown in Figure 10a and 10b, and an explanation of how they are used follow.

![Figure 10a. Multiplier A](image)

//used to determine the length of a round
double multiplierA = Math.Pow( 0.5 , thisRound );
int roundLength = totalDays * multiplierA;

![Figure 10b. Multiplier B](image)

//used to determine from when to schedule a match depending on the round number
double multiplierB = Math.Pow( 0.5 , thisRound – 1 );
DateTime roundDateStart = startDate.AddDays( totalDays - ( totalDays * multiplierB ) );

The fundamental principle that multiplier A takes advantage of is that half of the teams in the previous round will not be present in the current round. This calculation takes half to the power of the round number, which indicates how many teams should be left every round, subsequently indicating how many matches need to be played in every round. With reducing numbers of matches played per round as the tournament progresses, the duration of each round shortens. This implies that the gap between matches also shortens. This needs to be considered every time a round is scheduled.

For instance, using the eight-team tournament example above, if the start and end dates are 01/01/2015 and 01/02/2015, the first round will be scheduled over 0.5^1 * 31 = 15 days, the second round will be scheduled over the next 0.5^2 * 31 = 7 days and the final round will be scheduled for the end date.

Multiplier B is used to determine where the previous round ended, so that the next round can be scheduled after this. It is best explained in practice, so it will be explained alongside the example above.

Determining when the first round will start is trivial; it is scheduled using multiplier 0.5^0 = 1, which implies roundDateStart = startDate.AddDays(0) = startDate. Round two is scheduled to start using multiplier 0.5^1 = 0.5, on roundDateStart = startDate.AddDays( [ 30 - ( 30 * 0.5 ) ] ) = startDate.AddDays( 15 ) = 15/01/2015, and the final is scheduled for the tournament end date.
These values correspond to the number of days in each round, as calculated using multiplier A.

\[ \text{Frequency} = ( \frac{\text{Length of Round}}{\text{Number of Matches per Round}} ) \]

The frequency of matches is a value that specifies how often matches should take place in a particular tournament. If the length of a round is 15 days and there are 8 teams playing, then there are 4 matches in that round, and thus a match is played every \( \frac{15}{4} = 3 \) days. The fixture list will then be as follows:

Match 1 – 02/01/2014
Match 2 – 05/01/2014
Match 3 – 08/01/2014
Match 4 – 11/01/2014

A further requirement as specified by Douglas, was that no more than one match for a particular tournament be played on any given day. Thus, if the frequency is less than zero, matches are automatically scheduled at a frequency of 1.

**ii. Opponent Selection**

Matchups are decided randomly, however, each team plays any other team at most once. This is the nature of knockout tournaments.

5.4.3. Round-robin Tournament Scheduling

A round-robin tournament is a tournament where every team plays each other once, and the winner is calculated as the team who won the most matches.

Unlike knockout tournaments, there are no regulations that dictate valid round-robin tournament fixtures.

**i. Scheduling**

An entire round-robin tournament is scheduled from the start date through to the end date. As with knockout tournament scheduling, helper values were identified and used in the scheduling of round-robin tournaments.

\[ \text{Number of Rounds} = ( n - 1 ) \]

The number of rounds in a knockout tournament can also be calculated given the number of teams. If there are \( n \) teams playing, there are a total of \( ( n - 1 ) \) rounds. This is because, since each team has to play every other team, if there are 8 teams in the tournament, then 1 team must play 7 matches i.e. 1 match per round.

\[ \text{Number of Matches} = \left( \frac{n \times (n - 1)}{2} \right) \]

The number of matches in a round-robin tournament is dependent on the number of teams playing. If there are \( n \) teams playing, then there are a total of \( \left( \frac{n \times (n - 1)}{2} \right) \) matches.
matches played altogether. For instance, if there are 8 teams playing, namely team A, B, C, D, E, F, G and H, the first round will have 4 matches, the next round will have 4 matches, the third round will have 4 matches and so on, until all teams have played each other. This calculates to 7 rounds and thus, a total of 28 matches will be played.

\[ n \times (n - 1) / 2 \]

*Frequency* = \( \frac{\text{Length of Tournament}}{\text{Number of Matches}} \)

Match are scheduled at uniform intervals between the start and end date, and thus this period can be calculated as a function of the length of the entire tournament and the total number of matches in the tournament. Thus, if an 8-team tournament were to start on the 01/01/2015 and end on the 01/02/2014, matches would take place every \( \frac{31}{26} \approx 1.2 \) day.

**ii. Opponent Selection**

Figure 11 is an implementation of the round-robin scheduling algorithm. It iterates through a list of teams, manipulating the index to ensure that every team is matched up against every other team.

---

3 If frequency is calculated as less than 1, it is fixed to 1 to prevent same-day fixtures.
First, line A checks whether there is an even or odd number of teams. If there is an odd number, one team will have no match for that round, recorded as a “Bye”.

Line B removes the first team in the list creating a position to pivot around.

Line C creates a variable called teamIndex, which keeps track of all the matchups between the team that was removed and the other teams.

Lines D add teams to the matchup list called versus with the intention of listing competing teams, versus[0] vs. versus[1].

Figure 11. Round-robin Scheduling Algorithm
Line E is a loop that sets up the rest of the matches. Since a matchup between team one is organized for every match, there is \((n/2 - 1)\) matches that still need to be organized. Note, the \(-1\) in the expression above is accounted for as the iterator starts at 1, not 0.

Lines F and G calculate which teams remain to play. It uses the iterators of the inner and outer loops, using the modulus of the number of teams participating to wrap around the list of teams so that each team is matched up against every other team.

5.4.4. Dispute Resolution

There are two ways that disputes can be logged, automatically and manually. Disputes are logged automatically if players A and B upload a different set of match results for the same match. Disputes can also be logged manually if player A sees that the player B has uploaded an incorrect match result, but player A is not able to upload the screenshot at that point in time.

Regardless of how disputes are logged, they are resolved in the same manner. The administrator is able to view all of the disputes from the administrator tools tab and resolve specific disputes by checking individual dispute details, as shown in Figure 12. The details page shows an overview of the match, indicating each team captains email address and each team’s uploaded screenshot of the match result. There is a dropdown box at the bottom of the page that provides the administrator with an input to choose the match winner. Selecting a match winner will resolve the dispute between the two teams.

Figure 12. Screenshot of Dispute Resolution
5.5. System Workflow

The system works as follows. A new user registers himself on the system, choosing a username and password, along with other personal details. After his profile is created, he must set his availability window for each day. Once his profile is setup, he joins the league he wants to compete in, creates or joins a team and that team joins a tournament.

Before the tournament is scheduled to start, the administrator creates a fixtures list by going to administrator tools, then manage tournament, after which, if all the requirements for the tournament to start, he can schedule the tournament.

The tournament is then scheduled according to the players’ availability windows, and the administrator has the ability to reschedule matches.

As matches within a tournament are played, players upload results. If there is a discrepancy between uploaded results, a dispute is logged. The administrator can view and resolve disputes under manage disputes, in administrator tools.

Match results are stored in a table, and these are used to determine either who goes into the next round (knockout tournaments) or who won the most matches (round-robin tournaments).

The database schema is shown in the appendix <Appendix D>.
6. Testing Phase

During the final sprint of the project development life cycle, the eSports Portal was tested thoroughly for accuracy, performance and errors. There are three components that make up the testing phase; functional testing, non-functional testing and exception handling. The following sections discuss these tests as they apply to our project.

6.1. Functional Testing

Functional tests relate to testing the use-cases of the system, ensuring that these use-cases are met fully and accurately.

6.1.1. Scheduling Accuracy

The most complicated methods in the administrator component of the project were the scheduling methods. An overview of the algorithm is as follows:

1. Schedule match dates using tournament end and start dates
2. Get the teams involved in the tournament.
3. Get the availability windows.
4. Pair teams up in matches.
5. Find availability overlaps on match dates, considering differing time zones.
6. Schedule match on match date at either available overlap time or 00:00a.m.

Establishing that the generated fixture list is accurate involved confirming the following criteria:

a. The teams that were extracted in [2] are actually a part of that tournament.
b. The availability windows compared in [5] correspond to the teams that were paired up to compete in [4].
c. The day in the overlaps availability window in [5] was being compared to the day of the scheduled match date in [1].
d. Time zones were correctly accounted for.
e. The tournament rules were correctly applied i.e. each team played every other team exactly once for round-robin; while all winners and no other teams proceeded to subsequent rounds in knockout tournaments.

Once these criteria were met, the resulting scheduling system would accurately schedule matches according to a player's availability and local time zone.

---

4 00:00a.m. is only used in testing, to determine when finding an overlap was unsuccessful. The system uses a value of 13:00 as a default, if no common availability window was found.
6.1.2. Dispute Resolution
The fundamental concept in dispute resolution is correctly extracting the relevant match information pertaining to that match dispute. The administrator should be presented with the match results uploaded by each team, with the teams’ captains contact information displayed for ease of reference. Verifying that these details were correct implied that disputes and their details were being accurately presented. The administrator would then choose a winner, thus resolving the dispute. Verifying that the dispute status changed to resolved and that match winner correctly changed in the relevant tables, indicated that the dispute resolution functionality was accurately implemented.

6.2. Non-functional Testing
Non-functional tests on the other hand are performed to establish that non-functional requirements such as scalability and usability requirements are adequately supported.

6.2.1. Scalability, Performance and Load Testing
Simulating a real-world test environment by running the system on Derivco’s servers and accessing it from a variety of place around the world would be the ideal way to tune and test these non-functional requirements. However, due to limited resources, we were unable to simulate such an environment, and had to fall back on the next best thing; a locally run Web server with hypothetical test data.

Visual Studio enables test data to be created using code through a seed method, which was used to fill up the database with test users, test leagues and test tournaments. The project was tested with over one thousand test users and ten test leagues with four test tournaments per each test league. The system was able to perform all necessary functions without any noticeable delay, and thus the system was deemed scalable with adequate performance.

It must be noted that this test gives no insight into the performance when there are multiple users using the system at once. It also does not show how the system would perform if the Web server is located in a different region to that of client. These issues will be resolved should Derivco decide to use our system.

6.2.2. Integration Testing
Integration tests ensure that all the separate components and layers cooperate together to satisfy a request. These tests needed to be run after every merge between the administrator component and the player component. Testing involved testing each feature separately, and then together, to determine if the outcome is the same. Because the components of the project were mostly independent of the other components, the focus of integration testing was on ensuring that the project would run after the merge.
6.2.3. Security Testing
The security of the system is enforced through two mechanisms, password-based authentication and role authentication. The password-based authentication is used to authenticate users, and was tested by trying to log into an account without the correct credentials. This test was successful, as the system prevented the user from logging in. Role authentication is used to control access-privileges. It was tested on one Web page with the role authentication syntax included in the code. It was deemed that if this Web page correctly redirected a user without sufficient access to a "No Access" page, it would do the same for other restricted URLs. This test was also successful.

6.3. Exception Handling
The final component, exception handling, ensures that any error that is thrown is accompanied with a helpful message that informs the user of what caused the error and how it should be fixed. The following is a list of exceptions that were handled, categorized by the feature they are a part of.

Table 2. Error Cases

<table>
<thead>
<tr>
<th>Feature</th>
<th>Error 1</th>
<th>Error 2</th>
<th>Error 3</th>
<th>Error 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>When a user wants to choose a username that already exists</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leagues</td>
<td>When a user wants to create a league with a name that already exists</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tournaments</td>
<td>When a user wants to create a tournament with a name that already exists</td>
<td>When a tournament is created with a start date that is after the current date</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Scheduling</td>
<td>No availability overlap found for a particular match date</td>
<td>Schedule a round in a knockout tournament while there are unresolved disputes</td>
<td>Schedule a round when not all teams in the previous round have played</td>
<td>Scheduling a tournament when the current teams in the tournament is less than then number of teams required for the tournament</td>
</tr>
<tr>
<td>Disputes</td>
<td>When a dispute is logged without any match results being uploaded</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
7. Conclusion

7.1. Summary
The objective of this project was to build an online application to manage eSports tournaments on a global scale. This report contains detailed information of how the system was built, pertaining specifically to the administrative features.

The project was developed over nine two week sprints, with three sprints making up one sprint cycle. A break down of what was covered in each sprint cycle is as follows:

Sprint Cycle One
- Introduction to the Microsoft .NET framework
- User management implementation
- League management implementation

Sprint Cycle Two
- Tournament management implementation
- Scheduling design

Sprint Cycle Three
- Scheduling implementation and testing
- Dispute resolution design, implementation and testing
- Non-functional testing

The application was built using the Microsoft .NET framework in Visual Studio 2013, which required us to learn new technologies such as Microsoft MVC framework 4.5, C#, HTML and CSS. After learning these technologies and frameworks, requirements were extracted from a representative of the client company and the project was designed to meet these needs. Implementation of the designed features then began, and the project came to its conclusion with functional and non-functional testing.

7.2. Reflection
At the start of the project, while learning the ropes of the Microsoft .NET framework, it was intimidating that I knew so little about the development language and environment that we needed to use for the required system. The only thing I took comfort in was that the Java and C# were very similar languages, syntactically. After doing the tutorials introducing the .NET framework with MVC a few times, my understanding grew tremendously, and I was able to start development competently. Any issues that I ran into, from indiscernible error messages to useful functions that I was not aware of, I was able to find a solution either online or through other students who were familiar with Visual Studio. I found that the programming foundation that I built...
throughout my academic career was sufficient enough for me to thrive in learning an entirely new programming framework.

From a software engineering perspective, this was also my first time using a collaborative online project management tool to plan out the product backlog. I believe that if I were to do this project differently, I would pay more attention to the product backlog deadlines, and track work done on the project as it is completed. Following this closely is critical to ensuring that features are developed on schedule, and that each team member is doing an adequate amount of work on the project.

Overall, I feel that the project was completed successfully. My group members were able to help out when needed, and they were always approachable if I had any concerns regarding my functionality, or theirs.

I am truly amazed at how much I have learnt over the course of this project. Not only did I learn new skills, I grew to understand why these skills were important. These are not limited to technical skills. I was introduced to baiting requirements out of the client over a distance, I found myself improving on my organization skills and most importantly, I was able to appreciate the theory that I have been learning in the recent years, and apply it in a real-world scenario. If I could start the project over, the one thing I would do differently is record everything I have learnt in a personal notebook, as the learning took place.

7.3. Evaluation
The resulting product meets all of the requirements that the client expected. The project lifecycle was planned adequately and followed sufficiently enough that the project was completed in the designated amount of time, with all of the requested functionality.

7.4. Future Work
There are infinite possibilities to extend the core system that we have developed into something more than just an eSports management portal. I have included in this report a few features that I think would take our system to the next level in the foreseeable future.

7.4.1. Additional Social Information
The point of this system is to help Derivco build a rapport with employees in their newly acquired overseas offices. It would thus be beneficial if profiles were more personal so that the people from Derivco can learn more about the new office cultures that they have partnered with. This could be anything from an “About Me” page to a profile picture, something small to make it easier for Derivco and the other offices to learn something about each other.
7.4.2. Logging Scheduling Disputes
The current system only has the ability to log and resolve disputes that occur over match results. Scheduling disputes will allow users who are unable to play a match on a certain day to log a schedule dispute that indicates a better time that they can play. Opponents can then either confirm this new time or opt for a different time. If opponents cannot agree on a time, then the administrator will set a neutral date and time.

7.4.3. Customizable eSports News Feed
There is currently an RSS reader built into the system to extract news from an eSports news Website. The issue with this is that irrelevant, unrelated news articles pop up, which is detrimental to the use of the system as it slows the system down. In the next version, there should be an internal news feed that is developed and maintained by IT support, so that news articles are informative, current and relevant to the users.

7.4.4. Tournament Type Variety
There are only two types of tournaments currently supported by the system, knockout and round-robin. In the future, this is going to change. In the design phase of the project, we thought of two additional tournaments types we would like to see the system cater for, World Cup and double elimination.

A World Cup tournament follows the same structure as the football world cup, in that first teams are split up into groups. Each group plays a round-robin within that group, and the top two teams from each group go on to the knockout stages (quarter-final, semi-final and final).

Double elimination is similar to knockout, but instead of being knocked out after losing one match, players are knocked out only after losing two matches.
8. Bibliography


# 9. Appendix

## 9.1. A: Sprint Cycles Two and Three Overview

<table>
<thead>
<tr>
<th>Task Description</th>
<th>Story Points</th>
<th>Epic Points</th>
<th>Status</th>
<th>Sprint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a tournament</td>
<td>5</td>
<td>5</td>
<td>Done</td>
<td>Sprint 4</td>
</tr>
<tr>
<td>Create a league</td>
<td>5</td>
<td>5</td>
<td>Open</td>
<td>Sprint 4</td>
</tr>
<tr>
<td>Designing Scheduling Systems</td>
<td>50</td>
<td>30</td>
<td>Open</td>
<td>Sprint 5</td>
</tr>
<tr>
<td>Dispute Resolution Strategies Design</td>
<td>50</td>
<td>50</td>
<td>Open</td>
<td>Sprint 6</td>
</tr>
<tr>
<td>Implement Scheduling System</td>
<td>50</td>
<td>50</td>
<td>Open</td>
<td>Sprint 6</td>
</tr>
<tr>
<td>Complete Scheduling System</td>
<td>20</td>
<td>20</td>
<td>Open</td>
<td>Sprint 7</td>
</tr>
<tr>
<td>Combining Scheduling System into MIS Project</td>
<td>10</td>
<td>10</td>
<td>Open</td>
<td>Sprint 7</td>
</tr>
<tr>
<td>Document Dispute Resolution Strategies</td>
<td>20</td>
<td>20</td>
<td>Open</td>
<td>Sprint 7</td>
</tr>
<tr>
<td>Incorporate Timezones into Availability</td>
<td>20</td>
<td>20</td>
<td>Open</td>
<td>Sprint 8</td>
</tr>
<tr>
<td>Refine Use Cases</td>
<td>50</td>
<td>50</td>
<td>Open</td>
<td>Sprint 8</td>
</tr>
<tr>
<td>Design Class Diagrams</td>
<td>40</td>
<td>40</td>
<td>Open</td>
<td>Sprint 8</td>
</tr>
<tr>
<td>Implement administrator dispute resolution</td>
<td>10</td>
<td>10</td>
<td>Open</td>
<td>Sprint 9</td>
</tr>
</tbody>
</table>
9.2. B: CalculateOverlap Method

// Compare two AvailabilityWindows to determine a Common Availability Window
// Returns the Common Availability Window
public static AvailabilityWindow calculateOverlap(AvailabilityWindow p1,
 AvailabilityWindow p2)
{
    AvailabilityWindow commonTimes = new AvailabilityWindow(0, "", "", new bool[6,48]);
    AvailabilityWindow p1temp = new AvailabilityWindow( p1 );
    AvailabilityWindow p2temp = new AvailabilityWindow( p2 );

    bool flag = false;

    for (int rows = 0; rows < 6; rows++)
    {
        for (int columns = 0; columns < 48; columns++)
        {
            if (p1temp.timeSlots[rows, columns] &&
                p2temp.timeSlots[rows, columns])
            {
                commonTimes.timeSlots[rows, columns] = true;
                flag = true;
            }
        }
    }

    if (flag)
    {
        return commonTimes;
    }
    else
    {
        return commonTimes; // send conflict email
    }
}
9.3. C: GetOverlaps Method

```csharp
// Iterates through Availability Window
// Returns the Common Availability Window Options as a String Array
public List<string> getOverlaps()
{
    var option = new List<string>();
    bool flag = false;

    //outer loop for days
    for (int rows = 0; rows < 6; rows++)
    {
        //inner loop for times
        for (int columns = 0; columns < 48; columns++)
        {
            if (this.timeSlots[rows, columns])
            {
                option.Add((rows) + "t" + (columns - 8));
                flag = true;
            }
        }
    }

    if ( !flag )
    {
        option.Add("-1 -1");
    }

    return option;
}
```
9.4. D: Database Schema