

Software Development Fundamentals

Project Skills and Professionalism

Algorithms & Constructs

RUGBY CLUB SYSTEM

STUDENT: DIANA CABEZAS. STUDENT #: 2023073. DATE: 15.11.2023

Table of Contents

[Introduction: 3](#_Toc151140466)

[Software Development Fundamentals: Modelling and Testing 3](#_Toc151140467)

[UML - Use cases for a Rugby club System: 3](#_Toc151140468)

[UML – Data Structure Diagram: 5](#_Toc151140469)

[UML – Class Diagram: 6](#_Toc151140470)

[Algorithms and Constructs 7](#_Toc151140471)

[Hardware Overview: 7](#_Toc151140472)

[Sort Algorithm: 7](#_Toc151140473)

[Why did I choose Selection sort and not an alternative? 7](#_Toc151140474)

[Search Algorithm: 7](#_Toc151140475)

[Why did I choose Linear Search and not an alternative? 7](#_Toc151140476)

[Project Skills and Professionalism 8](#_Toc151140477)

[Problem: 8](#_Toc151140478)

[The best practices and solutions in software design: 8](#_Toc151140479)

[Data Modeling Best Practices: 10](#_Toc151140480)

[Definition card for Rugby Club System 11](#_Toc151140481)

[Four-Step Approach 12](#_Toc151140482)

[Risk assessment of the Rugby club system: 14](#_Toc151140483)

[Conclusion: 16](#_Toc151140484)

[Bibliography 17](#_Toc151140485)

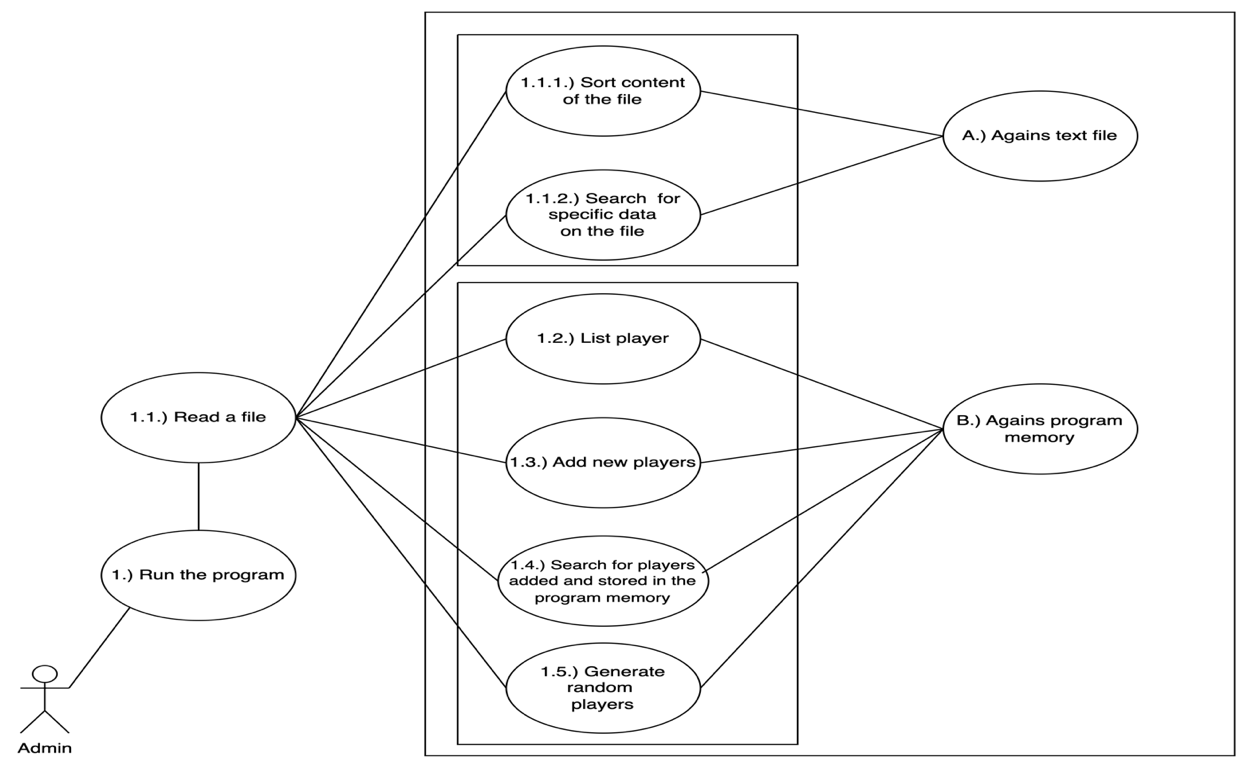
# Introduction:

We have been tasked to model and create a program to demonstrate a “Rugby Club” as a protype, this will be built in java programming language. The purpose of this program is to allow the user “admin” to add players and assign them a coach and a team depending on the skill each player has. This System will help keep track of the players on a team and the coaches they have in the rugby club, you will also be able to see a list of applicants who want to be part of the club. This document will content an explanation of the 3 different kinds of UML modelling techniques used to create the model of this software system “Rugby Club” along with an explanation of the unit tests implemented. This also will talk about the problems associated with designing the Rugby Club program, and the solutions taken to address them.

# Software Development Fundamentals: Modelling and Testing

It's our job to model three different kinds of UML modeling techniques for our Rugby Club software system. We're going to talk about the different UML techniques we used to model our Rugby Club system. To do this, we implemented UML - Use Cases, Data Structure Diagrams, and Class Diagrams. I'll show and explain each one below.

## UML - Use cases for a Rugby club System:



This diagram represents the use cases of our System and reflects the way our admin interacts with it. For the actor to use the system he needs to run the program and after this read the text file to have access to all functionalities, if the user does not provide an existent text file, it won’t let the user use it. Without further ado, let's break this down!

**Actor:** As you can see our actor here is the administrator: Why is our UML directed only to admin user? It is worth remembering that the System will maintain a record of the players that have been entered by the user and stored in the memory of the program, this will also allow the user to read a text document that contains a list of names of applicants, these are functionalities that only one specific actor should have, that is why I assume that this system will only be used by the administrator of the rugby team.

|  |  |
| --- | --- |
| *Uses cases* | *Description* |
| 1. Run the program | The first ellipse shows the action the actor has to perform in order to use the application, once this is done the user will be able to use the functionalities our system has. |
| * 1. Read txt file | This ellipse shows that the user can read a content of a file “Club\_Form.txt”. |
| * + 1. Sort content of the file | In this ellipse express that user can sort the data in this case it will be sorted in alphabetical order. |
| * + 1. Search for specific data on the file | This ellipse informs that the user can search for data from the file, here the user can search for a name of the player and get it along with his information and display it in console. |
| * 1. List players | This will list the information of the players added and stored in the program memory. |
| * 1. Add new players | This ellipse shows the functionality of adding a new player and store it on the program memory. |
| * 1. Search for players stored in the program memory | This will search for a player from the list store in the program memory, you can search for a player by the name. |
| * 1. Generate random players | This will get a player randomly from the list store in the program memory. |
| A | Ellipse A has been created to represent which functionalities can be apply against the text file we are reading. |
| B | Ellipse B has been created to represent which functionalities can be apply against the list store in the program memory. |

## UML – Data Structure Diagram:

In the specifications of our System, it says that it will have three classes and each of them could contain three subclasses. In my System I implemented three parent classes which are Player, Coach, and Team. The player class has no child class, but the coach and team classes have three subclasses each. Why have I chosen this model? Well, I believe that my system is aimed at recruiting players who will be beginners and there is no need to classify them since each player will be assigned a coach depending on his level. This data structure diagram is a visual and clear representation of our system and how the classes are connected. This shows the interface, the implementation, the three classes and each of their subclasses.

A black background with white rectangles

Description automatically generated

Here is an explanation of what each rounded square represents:

**R.C. Interface:** Here is where I am going to define the various menu options and list all the methos I will use.

**R.C. Class implementation:** This one will content all the implementation code for the method of my classes.

**Class-Player:** This class is to create more objects of the player class and it will have the following parameters:

Player (String name, String last\_name, String email, String gender, Coach coach, Team team); as you can see this class takes as parameter a Coach and Team object.

**Class-Coach:** This class contain three subclasses (Head Coach, Assistant Coach, Scrum Coach) thay all extend from this class, and they will take (String name) as a parameter.

**Class-Team:** This class contain three subclasses (A Squad, B Squad, Under-13 Squad) they all extend from this class, and they will take (String name) as a parameter.

## UML – Class Diagram:

By using our club rugby system, you will be able to:

* The players should be listed.
* New players should be added.
* Randomly generate players.

As a result, we show the methods applied to our classes in the diagram below as well as the contents of our interface.

A black background with white text

Description automatically generated

This class diagram describes the structure of our Rugby Club System showing the classes, attributes, and the relationships among objects. The Player class is formed by 6 attribute (**name, last\_name, email, gender, team, and coach**) the last two variables are Coach and Team type which means It will take COACH and TEAM object as a parameter.

Then we have the class Coach that take a String as parameter and have the subclasses (**Head Coach, Assistant Coach, and Scrum Coach**) these classes extend from Coach class and will etherate properties from it, same structure applies for the Team class with its respective subclasses.

# Algorithms and Constructs

## Hardware Overview:

Model Identifier: MacBookPro12,1

Processor Name: Dual-Core Intel Core i5

Processor Speed: 2.7 GHz

Number of Processors: 1

Total Number of Cores: 2

L2 Cache (per Core): 256 KB

L3 Cache: 3 MB

Memory: 8 GB

## Sort Algorithm:

We have been provided with a list of random names in a file called **Club\_Form.txt** gathered from the form submissions of interested applicants to the Rugby Club, the administrator of the team wants us to order this list in alphabetic order.

To do this we have used the Selection sort algorithm. The time complexity of Selection Sort is O(N2). (Gosh, 2023)

## Why did I choose Selection sort and not an alternative?

Selection sort is easy and simple to understand and works well with small datasets like the file provided, with this size of data its time complexity would not be a big issue and can provide a simple and efficient solution.

As you can see in the overview of my hardware previous shown, its memory and processing power are limited Selection Sort can be implemented in little amount of code which makes it a good choice me.

## Search Algorithm:

We Also have been asked to search in the List and return relevant information. For this I have used the searching method Linear Search.

## Why did I choose Linear Search and not an alternative?

I picked Linear search because it's a simple algorithm easy to understand and implement, because we don't have a lot of data. So, if you have got a lot of data, you will need another search method because linear search will check each element one at a time, the larger the data, the longer the search will take.

You can do linear search on unsorted data; it doesn't have to be sorted. Furthermore, this doesn't require any extra memory, which is great for me.

# Project Skills and Professionalism

Problem: This is a small rugby club that has 3 coaches: Head Coach, Assistant Coach, and Scrum Coach. Likewise, the club has three teams: A Squad, B Squad, and Under 13 Squad. When this rugby club started operating it had very few recruits which made it very easy to keep track of the players, lately the club is growing in popularity and has a large list of applicants who want to register and who he would like to add. The club administrator wants to be able to add new players and assign them a coach and a team at the same time, the administrator would also like to be able to list the players that have been added with their respective coach and teams and select random players. He would also like to be able to see the list of all applicants that have been stored in a text file.

## The best practices and solutions in software design:

A developer can follow tons of best practices when it comes to software development. Developers should consider practices such as reducing code complexity, agile development, code review, etc. Although there's no universal practice, it depends on the project requirements as well. (Bhatt, 2022)

1. **Create an SRDS:**

* In a Software Requirements and Design Specification document, or SRDS, the requirements of the project are clearly defined, this simplifies the process of aligning resources with tasks.
* The document includes software design characteristics, design diagrams, decisions, constraints, and much more. Developers can easily figure out what the project demands with the SRDS. (Bhatt, 2022)

1. **Keep the code simple:**

* Make sure you follow the KISS (Keep It Simple, Stupid) principle.
* Organize complex tasks into smaller, manageable parts.
* Name your variables and functions in a meaningful way.
* Utilize design patterns when appropriate and avoid unnecessary complexity.
* It is easier to read, maintain, and debug code when it is simple. (Bhatt, 2022)

1. **Have proper documentation:**

* If the logic of the code is not immediately apparent, document it using comments.
* Provide administrators and coaches with user manuals and guides.
* Update the README file with instructions on how to set up and run the application.
* Documentation helps developers and users onboard efficiently. (Bhatt, 2022)

1. **Focus on agile development:**

* Embrace an incremental and iterative development process.
* Deliver features in short, frequent cycles based on importance.
* Review and adapt the system regularly based on feedback from coaches, administrators, and other stakeholders.
* Responses to changing requirements can be made more flexible with agile development. (Bhatt, 2022)

1. **Always remember unit testing:**

* Unit tests should be written for each component and function.
* Test different scenarios, including edge cases, with test cases.
* Make sure that changes or additions to the codebase don't introduce new bugs by using automated testing tools.
* Early detection of issues in the development process is made possible by unit testing. (Bhatt, 2022)

1. **Consider software maintenance beforehand:**

* To accommodate future growth in the number of players and coaches, design the system with scalability in mind.
* Create a modular and flexible codebase to facilitate updates and patches.
* To identify and resolve issues quickly, implement error logging and monitoring.
* Ensure security and stability by regularly updating dependencies. (Bhatt, 2022)

1. **Conduct a thorough code review:**

* Code reviews should be encouraged before changes are merged.
* Ensure that code adheres to coding standards and best practices in addition to checking for correctness.
* To improve the overall quality of the code, provide constructive feedback.
* The purpose of code reviews is to catch potential issues and promote knowledge sharing within the team. (Bhatt, 2022)

Our rugby club can develop a robust, maintainable, and scalable player management system by incorporating these best practices.

## Data Modeling Best Practices:

By planning how you're going to organize your data, you can improve performance, reduce errors, and reduce the chances of designing things wrong and having to rewrite code. Combining all these things should save you time and money. (Malher, 2023)

1. **Keep Business Objectives in Mind:**

* When modeling data, it is important to keep the overall business in mind, make sure you align your data model with reality by working with business analysts and other relevant stakeholders. As a software engineer, it can be easy to overlook the bigger picture. (Malher, 2023)

1. **Properly Document Your Data Model:**

* Your data model should be documented comprehensively, including entities, relationships, and attributes.
* Provide a clear explanation of each data element's purpose and use. Any assumptions or constraints that influenced the design should be documented. (Malher, 2023)

1. **Design Your Data Model to Be Adjustable Over Time:**

* Predict future changes and growth in the club's activities and players. Flexible data models can accommodate new requirements without requiring extensive rework. (Malher, 2023)

1. **Have Different Levels of Abstraction for Different People:**

* A high-level diagram can be used for stakeholders who need an overview, while a detailed model can be used for those involved in implementation. (Malher, 2023)

1. **Choose the Right Data Modeling Technique:**

* Choosing a data modeling technique that fits your system's complexity and requirements is important. UML diagrams and entity-relationship diagrams (ERDs) are common techniques. (Malher, 2023)

1. **Think about Data Governance and Security:**

* Implement data governance practices to ensure data integrity, quality, and compliance. (Malher, 2023)

1. **Avoid Premature Optimization:**

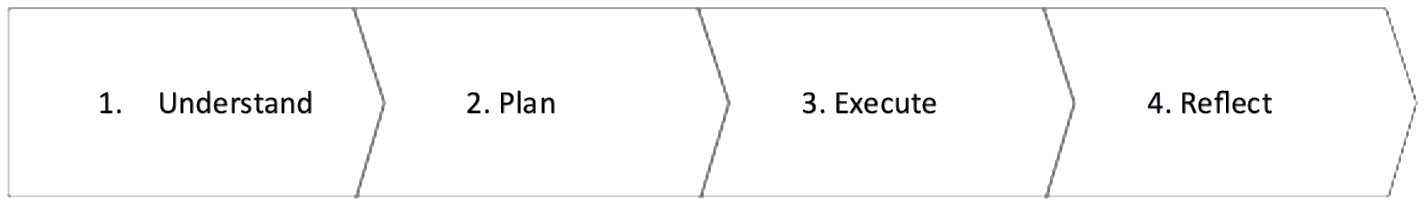
* Design a data model that meets the rugby club's current needs without being overly complex. (Malher, 2023)

## Definition card for Rugby Club System

|  |  |
| --- | --- |
| **Project Definition Card** | |
| **Problem Statement:**  The rugby club is experiencing growth in popularity, resulting in an influx of player applications. The club has three coaches (Head Coach, Assistant Coach, and Scrum Coach) and three teams (A Squad, B Squad, and Under 13 Squad). The club administrator needs a system to efficiently manage new player registrations, assign them to coaches and teams, maintain a player record, and view the list of applicants stored in a text file. | |
| **Features:** | |
| **Add new players** | * Allow the club administrator to add new players to the system. * Capture player details such as name, last name, age, and gender during registration. |
| **Assign Coaches and Teams** | * Provide the ability to assign each new player to a coach (Head Coach, Assistant Coach, or Scrum Coach). * Allow the administrator to assign players to one of the three teams (A Squad, B Squad, or Under 13 Squad). |
| **List Players** | * Display a list of all registered players with their respective coaches and teams. * Ensure that the list is easily readable and includes relevant player information. |
| **Random player selection** | * Implement a feature for the administrator to randomly select players from the program’s memory. * This could be useful for activities like forming random teams for training sessions or friendly matches. |
| **Applicant list from text file** | * Allow the administrator to view the list of all player applicants stored in a text file. * Ensure that the file reading process is efficient and error-handled. |
| **Additional Considerations:** | |
| **Scalability** | * Design the system to handle a growing number of players and applicants efficiently. |
| **User Interface** | * Create an intuitive user interface for the administrator to easily navigate through the system. |
| **Data Storage** | * Implement a secure and reliable way to store player information, considering future expansion. |
| **Benefits** | |
| * Streamlined player registration and assignment process. * Improved organization of player information for coaches and teams. * Quick and easy random player selection for various club activities. * Efficient retrieval of applicant information stored in a text file. | |
| **Success Criteria** | |
| * Successful registration and assignment of players without errors. * User-friendly interface for the administrator. * Accurate and up-to-date player lists. Proper handling of text file reading and error scenarios. | |
| **Constraints:** | |
| * Limited resources for system development. * Need for a secure and scalable data storage solution. | |
| **Stakeholders** | |
| * Rugby Club Administrator | |

## Four-Step Approach

In order to address the problem of managing three coaches and three teams in a growing rugby club, we are going to use the Four-Step Approach. This will help us develop, test, and refine the rugby club management system so that it meets the needs of both administrators and stakeholders.



1. **Understand:**

*Identify Stakeholders:*

* Rugby Club Administrator: The admin will manage player registrations, assignments, and all the functionalities our system has.

*Gather Requirements:*

* Player Registration: Capture player details during registration.
* Coach Assignment: Allow the administrator to assign players to coaches.
* Team Assignment: Provide the ability to assign players to teams.

*List Players:*

* List registered players with details on their coaches and teams in an organized and clear manner.
* Random Player Selection: The feature of random player selection should be implemented.
* Applicant List from Text File: Allow the administrator to view the list of all player applicants stored in a text file.

1. **Plan:**

System Architecture:

*User Interface:*

* Design an intuitive interface for the administrator.
* Database: To achieve better practices and to have backups, it would be adequate to have a database to store all the information the administrator will gather from the players.
* File Handling: Develop functionality to read the applicant list from a text file.
* Random Player Selection Algorithm: Define an algorithm for random player selection.

*Development Strategy:*

* Modular Design: Implement modular components for player registration, coach/team assignment, and list display.
* Iterative Development: Make sure features work and catch potential issues early by developing and testing incrementally.
* User Feedback: Gather feedback from the administrator during development to make necessary adjustments.

1. **Execute:**

Implement Features:

*Player Registration:*

* Create a form to capture player details.
* Store player information.

Coach and Team Assignment:

* Develop functionality for the administrator to assign coaches and teams to players.
* Update the list with assignment details.

*List Players:*

* Design a clear and user-friendly player list interface.
* Retrieve player information from the program memory and display it.

*Random Player Selection:*

* Implement an algorithm for random player selection.
* Provide an interface for the administrator to initiate random selection.

*Applicant List from Text File:*

* Develop a file reading mechanism to extract applicant information.
* Display the list of applicants to the administrator.

1. **Reflect:**

*Testing:*

* Conduct thorough testing for each feature to identify and fix bugs.
* Ensure data consistency and integrity.

*User Feedback:*

* Gather feedback from the rugby club administrator on the usability and effectiveness of the system.
* Make any necessary adjustments based on user feedback.

*Documentation:*

* The architecture and codebase of the system should be documented, along with any other information that will facilitate maintenance.

*Continuous Improvement:*

* User feedback and evolving requirements should be considered for future enhancements.

## Risk assessment of the Rugby club system:

1. **Defining the problem and the risks:**

A diagram of a diagram

Description automatically generated

*Image #1: Fishbone diagram risk assessment*

1. **Problems and the solutions taken:**

|  |  |  |  |
| --- | --- | --- | --- |
| Problem | Mitigation | Solution | Mitigation |
| *Technical Risks:*  Incompatibility with future technologies or dependencies. | * 1. Regularly update and maintain dependencies and design the system with flexibility to adapt to evolving technologies. | Regular Backups: | Implement regular automated backups to prevent data loss in case of system failures or errors. |
| *User Adoption Risks:*  Resistance from coaches and administrators to adopt the new system. | Provide thorough training, demonstrate the benefits, and gather feedback during the development process to address concerns. | Training and Support: | Provide comprehensive training sessions for coaches and administrators. Offer ongoing support and documentation to address any issues or questions. |
| *Data Security Risks:*  Unauthorized access to sensitive player information. | Implement strong authentication mechanisms, encryption for sensitive data, and conduct regular security audits. | Collaboration with IT Security Experts: | Work with IT security experts to conduct regular security assessments and ensure that the system is up to date with the latest security measures. |
| *Scalability Risks:*  Inability of the system to handle a growing number of players and coaches. | Design the system with scalability in mind, regularly optimize performance, and use cloud-based solutions if needed. | Modular Design: | Design the system with a modular structure, making it easier to update, maintain, and scale without affecting the entire system. |
| *Operational Risks:*  System downtime or errors affecting daily operations. | Implement robust error handling, conduct thorough testing, and have contingency plans for system maintenance or unexpected issues. | Agile Development Practices: | Embrace agile development to address evolving requirements and quickly adapt to changes, minimizing the risk of delivering a solution that becomes obsolete. |

1. **Analyzing potential impacts of solutions taken:**

|  |  |  |  |
| --- | --- | --- | --- |
| Positive Impacts: | Negative Impacts: | Long-Term Impacts: | Financial Impacts: |
| * 1. - Improved efficiency in player management.   2. - Enhanced communication between coaches and administrators.   3. - Streamlined registration processes leading to increased club popularity. | * 1. - Initial resistance to change from coaches or administrators.   2. - Downtime during system updates or maintenance. | * 1. -Continued growth in popularity and success of the rugby club.   2. -Increased satisfaction and engagement among players, coaches, and administrators. | - Initial investment in development and training costs.   * 1. - Potential cost savings in the long run due to improved efficiency. |

We can proactively address challenges and ensure our player management system implementation is successful by identifying risks, discussing mitigation strategies, and considering potential impacts. It's essential to revisit and update this assessment as the project progresses.

## Conclusion:

After having done the risk assessment on my project it is very evident how much this contributes to the development of a robust system and capable of being updated if required, appropriate documentation plays a crucial role here since this will be a key point for:

* Identify the different risks that we could face in the implementation of the project.
* If this is very clear, it will be very easy for the team to understand if we want this project to be implemented on a large scale.

I have applied the four steps approach. They were fundamental to be able to understand the requirements of our project, and to plan its implementation.

# Bibliography

Bhatt, S. (2022, Sepetember 2). *7 Best Software Development Practices to Follow in 2023*. Retrieved from BoTree Technolofies: https://www.botreetechnologies.com/blog/best-software-development-practices/

Gosh, A. K. (2023, November 14). *Selection Sort in Data Structures*. Retrieved from ScholarHat: https://www.scholarhat.com/tutorial/datastructures/selection-sort-in-data-structures#:~:text=Small%20Data%20Sets%3A%20Selection%20sort,a%20simple%20and%20efficient%20solution.

Malher, C. (2023, June 22). *7 Data Modeling Best Practices*. Retrieved from THENEWSTACK: https://thenewstack.io/7-data-modeling-best-practices/