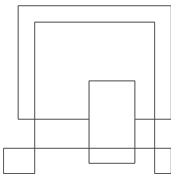


Project Report: Game Rental

Physical media distribution in the modern age

Christian Foyer 315200

Martin Rosendahl 315201

Levente Szajkó 315249

Kruno Nerić 315258

Supervisor: Steffen Vissing Andersen

Software Technology Engineering

2Y

May 23th, 2022

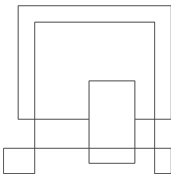
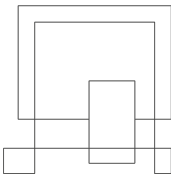


Table of content



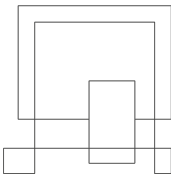
Abstract

To combat the increasing price of video games, a client server system has been developed to allow end users to rent games from a catalog that is managed by an administrator. The applications were developed in Java with the GUIs created using the JavaFX.

The database, which holds information about the users, games, rentals, and transactions, was implemented in PostgreSQL and was normalized to the third normal form to reduce redundancy. To allow renters to access the catalog, the remote model interface design pattern was implemented to connect to the host.

The result is a lightweight and easy to use application that has both user and administrator functionality. The renter can use the search function to look through the games for specific criteria, and then see their currently rented games on their profile. The administrator can in turn manage user permissions and update the inventory.

The system generates business value for the owners of the system by harvesting subscription fees from the user base for access to the service. In return, the renters can access a wide assortment of games for their preferred platform at a price that is far lower than purchasing each title outright.



1 Introduction

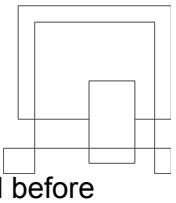
Video games form a large part of the modern entertainment business. With their ever-expanding genres and improving quality, video games have become a staple of indoor activities. Especially in the current day where outings have been replaced with playing games together over vast distances, with users from different regions of the world and different countries playing together. “The language of international communication is, of course, English” (Rudis, 2018). With a common language and game, these distances can be more easily bridged.

However with increased connectivity, came increased prices. According to Ben Gilbert of Business Insider, after nearly 15 years of stable prices for new video games, the cost has recently spiked (Gilbert, 2020). This caused buyers to spend large sums of money on games that they play for a limited amount of time and forget about soon after.

This leads to the second issue of buying video games, losing interest. As with all forms of entertainment, all video games will eventually lose the attention of players and be forgotten, and possibly never played again. The average game that is reviewed by end users is played for about 18 ½ hours and has a low average rating of 71% (Davis, 2022). With that in mind, the increase in price effectively reduces the overall value the consumers get from the product.

Comparatively, renting games is significantly cheaper than purchasing titles outright (Stegner, 2020). This allows video game consumers the option to try a wider variety of games for a lower price. However, there are advantages to buying a game, such as being able to play it whenever and as often as desired. It may not be worth the higher price considering many video game consumers spend less than 20 hours on a game (Davis, 2022).

This has become an even more active topic during the recent global health crisis, “During the ongoing period of lockdown and social distancing, many of us have observed a significant increase in the use of entertainment games (both analog and digital games) at home to pass the time and to deal with stress. Entertainment games are perceived as a relief for families, both for parents and children” (Kriz, 2020). Given



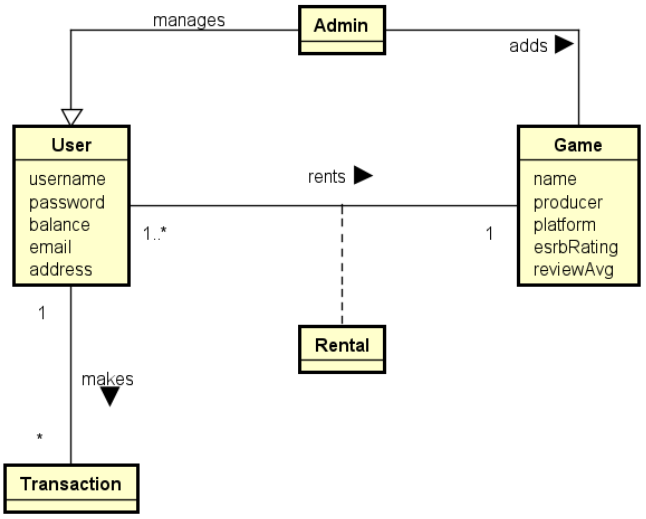
the increased demand for digital entertainment, the decreased value described before is striking.

Since english is used as the international language of communication (Rudis, 2018), the proposed catalog of games will be limited to english language games. In addition, to utilize the cost effectiveness of reusing physical copies, digital copies will not be offered on the rental service. This also avoids large legal issues with digital rights management.

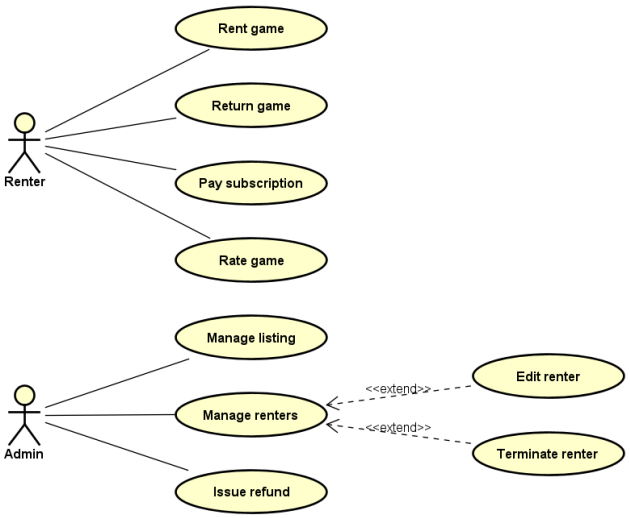
With the rental of games reducing the price, renting games can better harness the value (Stegner, 2020). For that reason, the game rental system was chosen as the goal of development. With that in mind, the following analysis section will discuss the domain of the problem and look at the requirements needed for a rental system.

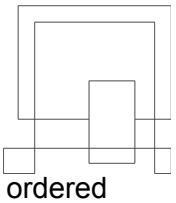
2 Analysis

The central interaction needed to generate business value for the stakeholders behind the system is the ability of the users to rent games. The other interactions within the domain of the problem aim to support that. For users to have games to rent, the games must first be added to the system. The administrator, which is a type of user, can take care of that as well as managing the users. So that other users may rent the games, they must also be able to return them. These actions that are key to the system should also generate a transaction to help keep track of what users are doing. With the small core of the problem domain defined, there is a clear path for the system to generate business value for the potential stakeholders.



To support the core interaction of the rental of a game, the different actors in the following use case diagram serve different purposes. As shown, the actor labeled as renter returns the rented games and then rates them. Since profitability is a core business value that the system should generate, the renter also must pay for their subscription. The administrator, labeled admin in the diagram, works to manage the inventory and the users.





From there, the user stories are transformed into functional requirements in an ordered list, starting with the highest priorities. After that, the non-functional requirements from the stakeholder are listed.

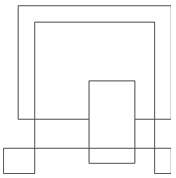
Functional requirements

Critical priority:

1. As a renter, I want to be able to rent games that I have access to for a specified amount of time, so that I can play them.
2. As a renter, I want to be able to search through listings by different categories, ratings, or name so that I can more easily find a game that I want.
3. As a renter, I want to be able to return rented games, so that I can rent new ones.

High priority:

4. As a renter, I want to login using a username and a password, so I can access the system on my own account on the system.
5. As a renter, I want to review my currently rented games, so that I can see if I can rent more games.
6. As a renter, I want to be able to register using my contact information including email, address, full name, date of birth, username, and password, so that I can have my own account on the system.
7. As an administrator I want to be able to add new items to the list so that the system displays up to date representation of the inventory.
8. As an administrator I want to be able to change information about the existing items so that the system displays up to date information.
9. As an administrator I want to be able to remove games that are no longer in stock so that the system only shows items that are in stock.



Medium priority:

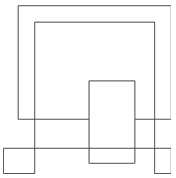
10. As an administrator, I want to be able to edit the renter information so they match the real world information.
11. As an administrator, I want to be able to terminate renters in order to prevent and punish fraud.

Low priority:

12. As a renter, I want to leave ratings for games I rented, to give feedback for other renters to view.
13. As a renter, I want to pay for my subscription and for any arising late fees,so that I can use the platform.
14. As a renter, I want to be able to see the balance on my account, so I can make payments.
15. As an administrator, I want to be able to issue a refund for an unsatisfied customer so that they want to continue using our services.
16. As an administrator, I want to be able to fine renters, so that I can discourage late returns.
17. As an administrator, I want to revoke renting privileges of renters, so that only trusted renters have access to the system.

Non-functional requirements

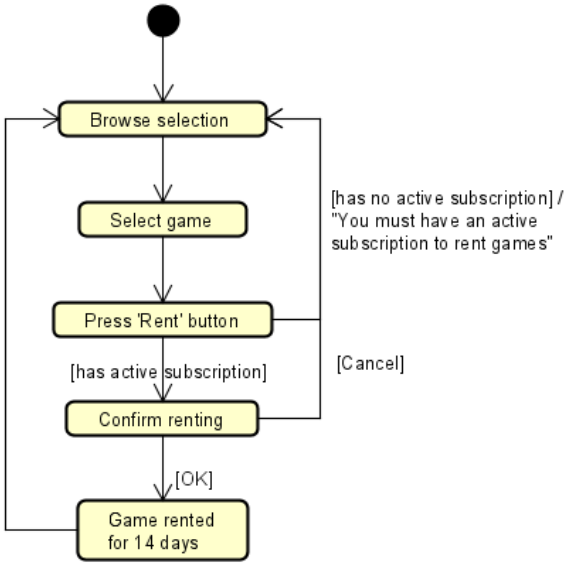
18. The renter must be 13 years of age or older.
19. Users' passwords are not stored in clear text.
20. User guides must exist for users and administrators to explain core functions.
21. The database must be able to drop the schema and create the tables again from scratch.



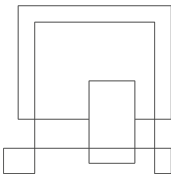
22. The client program must open within 60 seconds on a computer running Windows 10 and with a processor that has a clock speed of at least 1 GHz.
23. The database must be implemented in PostgreSQL so that the current system administrators can use their current knowledge base to maintain it.

Use Case	1. Rent a game
Summary	The Renter selects a game to rent
Actor	Renter
Precondition	The renter is registered and logged in to the system and has an active subscription.
Postcondition	A copy of the game is rented by the Renter and the process begins to send the physical copy to them.
Base Sequence	1. The renter browses the selection of games available for rent. 2. The renter selects the game they wish to rent 3. Confirm they would like to rent the selected game. 4. The game rental period begins on the game. 5. The game is added to the renter's list of rented games. 6. The game is shipped to the Renter's shipping address as specified in their profile. 7. The renter receives the physical copy of the game by mail. 8. The renter can play the game within the return period.

Branch Sequence	NO ACTIVE SUBSCRIPTION 1. The renter browses the selection of games available for rent. 2. The renter selects the game they wish to rent 3. Confirm they would like to rent the selected game. 4. Warning is given and the game is not rented
Exception Sequence	
Sub Use Case	Browsing the selection Viewing currently rented games
Note	Step 6-8 are handled externally

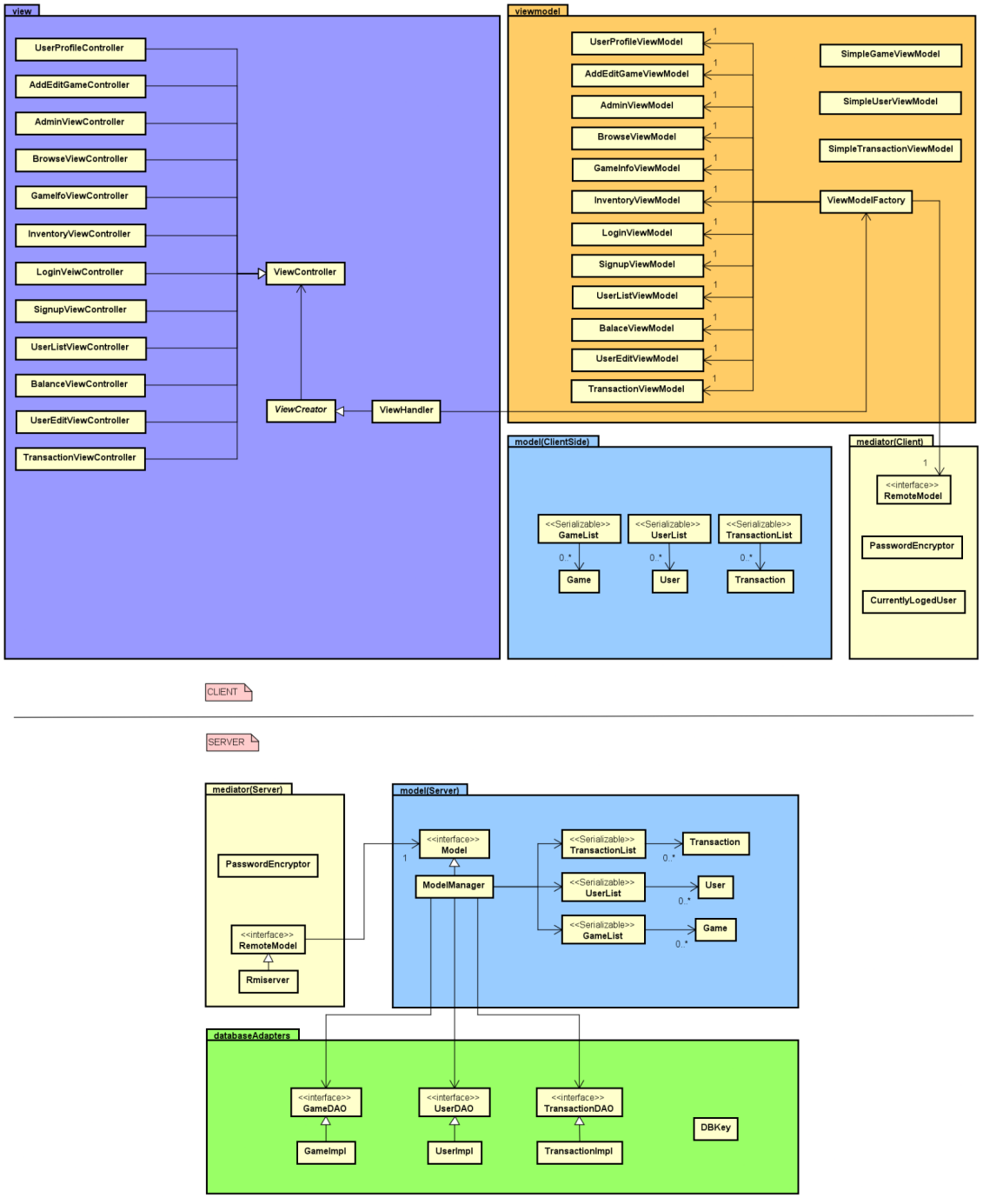


The highest priority requirement, which holds most business value and core purpose of the system is the ‘Rent a game’. Shown above is the fully dressed form of the use case. The use case holds multiple steps that require a real life person to handle shipping. On the left is the activity diagram of the same requirement, to help with visualization.



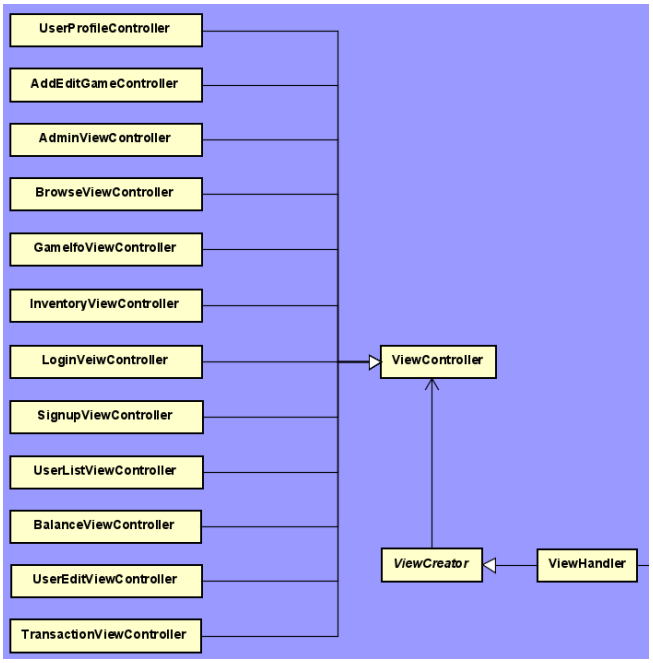
3 Design

To ensure proper functionality of the system, the core architecture was the client server split. It ensures that all clients have access to the same games. For this reason, having a single user system would not suffice since each user would have their own local copy of games and keeping track of which games are rented, and how many days are left in the rental period. To further improve functionality, readability and efficiency of the system multiple design patterns were utilized.



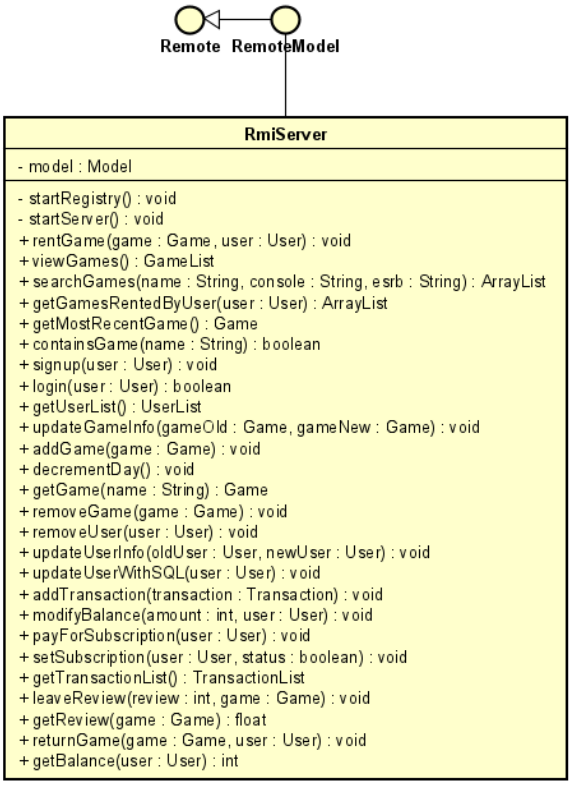
As it can be seen in the above shown simplified diagram of the source code, the first design pattern that was decided to be put in use was MVVM. This was done to improve the readability and segregation of classes, files and documents within the project.

MVVM also fits into the Single use requirement of SOLID principle, since it ensures that the functionality for the FXML files is not in the same class as its controls and on action events.



For controllers for the FXML files it was decided that the factory design pattern is going to be used. The purpose of this pattern was to ease implementation and altering of existing code when additional FXML files were required.

The factory design pattern was used in such a way that the *ViewHandler* does not need to be edited, no matter how many FXMLs and corresponding controllers there are. Whenever an FXML file was attempted to be opened the *ViewCreator* would locate the controller class for it before loading the FXML file itself.

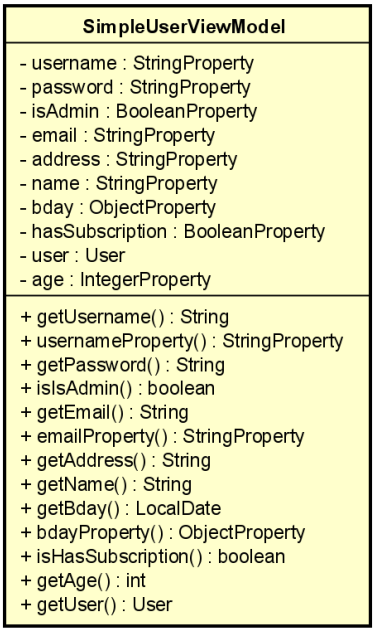


For the client-server split of the system it was unanimously agreed that RMI design pattern would be employed rather than manually defining sockets.

This was decided because RMI is much better at handling and manipulating objects, as opposed to sockets which need to have an object turned into a json string before it can be passed through the connection. Sockets while offering better control

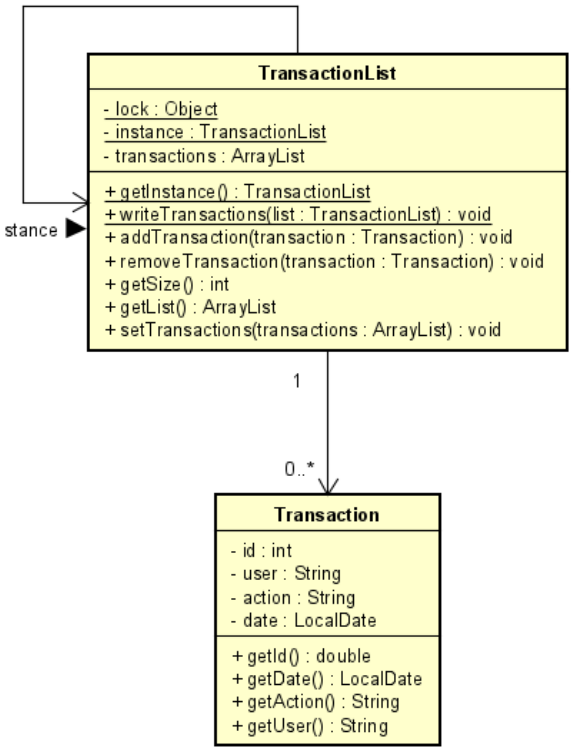
over the connection require an amount of scripted actions taken by the client and expected by the server that it makes them exceedingly difficult to use in the scope of this project.

One aspect of RMI that was left out is the callback. It was not found necessary because it was intended to be implemented in a way that stores added, changed or removed data to the database before the system reads it out and stores it locally.

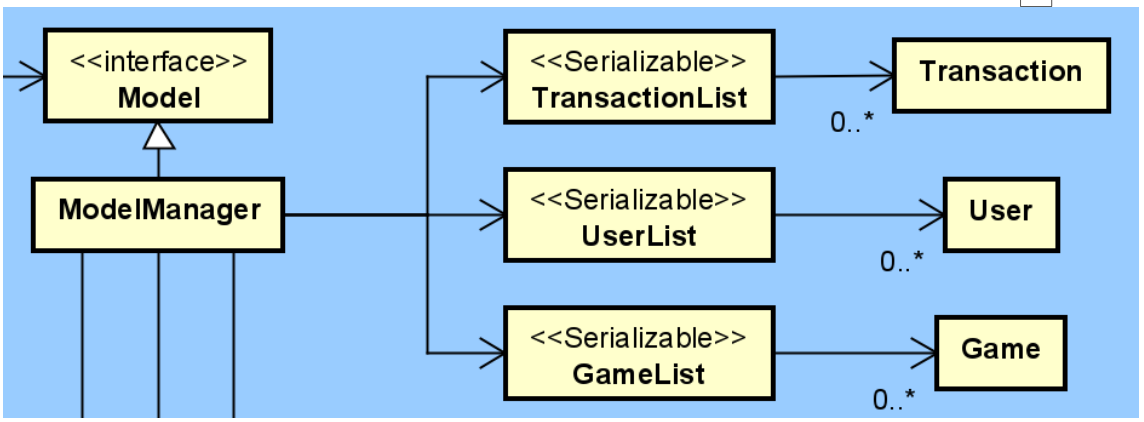


To make it possible to display data from objects inside tables it was necessary to create an adapter class for each class that needed to be shown in a table. For this purpuse adapter classes were created.

An example of an adapter is the *SimpleUserViewModel*. This class takes an *User* class object and transforms its attributes into their corresponding properties so that they may be presented inside a table.

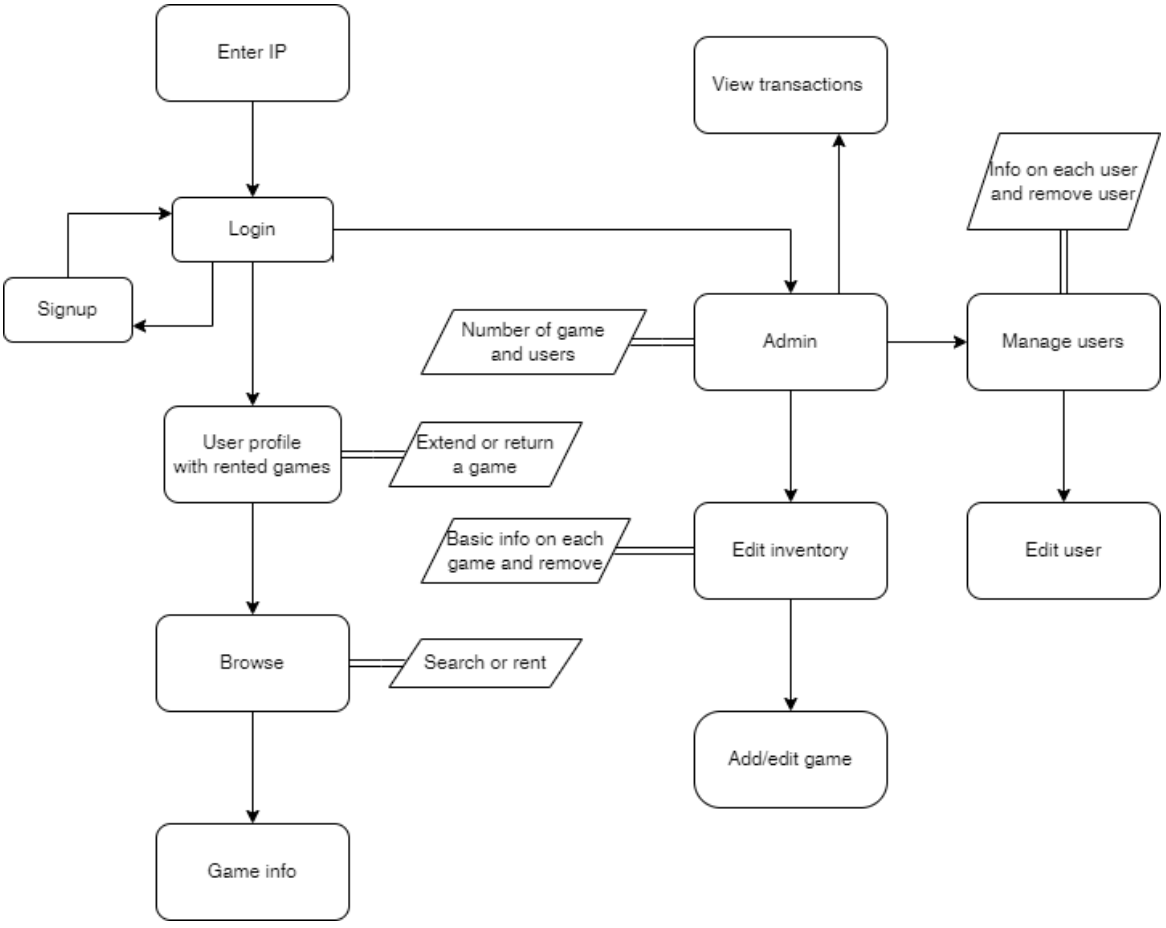


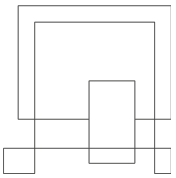
For keeping logs of renting and returning games as well as changing the account balance of the user, the singleton design pattern was used. This is also the class that keeps logs from the server. A Singleton design pattern was used because that made accessing the same logging object easier in every class



To allow RMI access to the model on the server side of the system, a façade interface was created. This allows the server to interact with transactions, users, and the games stored. In addition, it follows the decoupling principle within the SOLID design principles, which utilizes the associations between classes by delegating one level at a time.

The following diagram is a brief description of the navigation between windows when using the GUI. It demonstrates the flow of navigation to different views using the viewhandler class within the view package.





Upon starting the program, the user is first asked to define the IP address of the server that the user wishes to connect to. After filling out the field and confirming the input the user is taken to the screen requesting the user to input a username and password with which to log in. On this screen the user can log-in using their credentials, or create a new account with signing up.

If the user is already registered they fill out the fields and press the button to log in. If the provided information is correct the user is taken to their profile screen. they can also press the button to sign up where the user will once again be asked to fill out a form.

After filling out the form and pressing the sign up button they will return to the login screen provided that given information is acceptable. The user can also cancel the signup process at any point.

From the profile screen the user can go to their balance screen where they can manage payments and subscriptions, or go to the browse screen. If the user wants to they can also use the log out button which takes the user back to the login screen

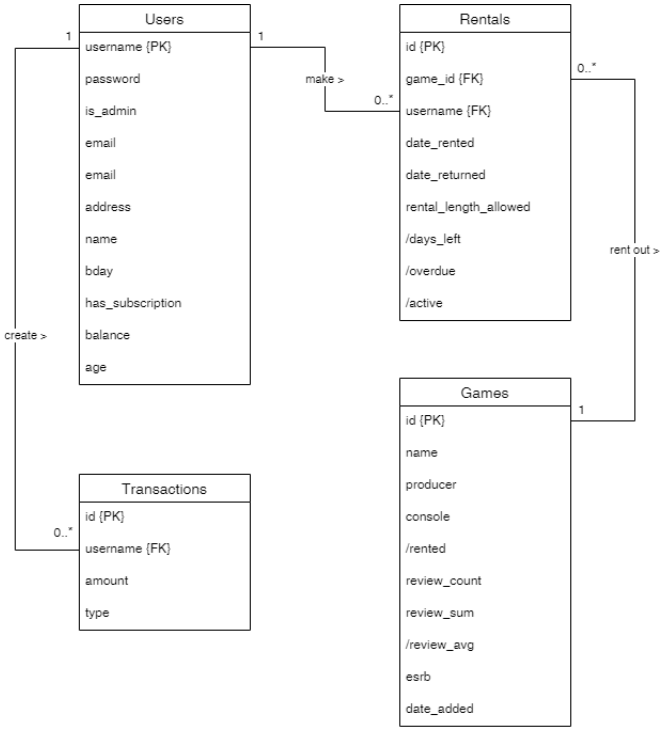
The browse window contains all available games the user can rent, as well as the option to see additional information about the game. The additional info is shown in a window that opens after a game is selected and the info button is pressed.

In the case the user logs in as an administrator they are taken to the administrator screen. On this screen, the user can navigate to a window for managing transactions, users and games as well as log out, which returns them to the login screen.

In the transaction management screen the administrator can view all transactions made by users and can navigate back to the administrator screen.

In the game management screen the administrator is able to add, edit and remove games as well as return to the administrator screen. Adding and editing a game leads the user to a smiliar screen in which the fields are empty or filled out depending if the user selected a game prior to pressing the button.

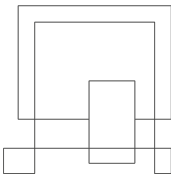
In the user management screen the program shows all the registered accounts in the system with the ability to change their informationIn the screen for editing a user's account the user can change the information given upon signing up, fine a user or give a refund or revoke that user's subscription.



For database design it was decided that four relations will be enough. Three containing basic data about users, games, and transactions, as well as one containing the game rentals created.

It was also found necessary to assign synthetic keys to games, transactions and rentals. A synthetic key was not needed for users because each username must be unique, which is stated in the signup process.

Triggers were employed for calculating age of users, days left for rented games, and decrementing the days left. This makes the database able to keep track of how long the user can have a specific game before they need to be returned and if the age of any user needs to be updated, so that the program can allow the user access to games with more mature themes based on the ESRB rating. Using these triggers makes maintaining the database easier, so that the administrator does not need to update the database everyday manually.



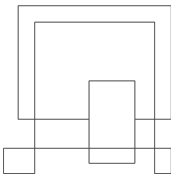
4 Implementation

The following section will show how some of the core features of the system have been implemented, how they function, and how they play into the previously described design patterns.

```
public void addGame()
{
    try
    {
        if (name.get().equals("") || producer.get().equals(""))
            throw new IllegalArgumentException("Name and producer cant be empty");
        Game gameToAdd = new Game(name.get(), producer.get(), console.get(),
                esrb.get());
        model.addGame(gameToAdd);
    } catch (Exception e)
    {
        error.set(e.getMessage());
    }
}
```

The process of creating and adding a new game to the database starts client side, in the AddEditGameViewModel. The method **addGame** (shown above) which creates the game that is to be added to the inventory first checks if the name and producers for the game have been provided. If the name and game have been provided, a new **Game** object is created using variables which are bound to corresponding fields and choice boxes in the GUI. The newly created object is then passed on to the server using the RMI interface **RemoteModel**, which is in the above code named simply **model**, or more specifically to the **addGame** method on the server.

```
@Override
public void addGame(Game game) throws SQLException, RemoteException
{
    model.addGame(game);
}
```



The server class, which implements the **RemoteModel**, simply passes the new game on to the model, to the identically named method of **addGame**. Such can be seen in the code above.

```
@Override
public void addGame(Game game) throws SQLException
{
    games.addGame(gameDAO.create(game));
    System.out.println("Game added: " + game.getName() + " on " + game.getConsole());
}
```

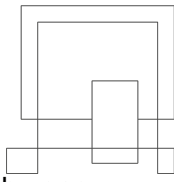
The game can only be added to the **GameList** after it has received an id from the database which will be used to uniquely identify the game. This is done using the **GameDAO** interface, whose purpose is to ast as an intermediary between the database and the system.

```java
@Override
public Game create(Game game) throws SQLException
{
    Game createdGame = null;
    try (Connection connection = getConnection())
    {
        PreparedStatement statement = connection.prepareStatement(
                sql: "INSERT INTO games"
                        + "(name, producer, console, rented, days_left," +
                        " review_count, review_sum, review_avg, esrb, date_added) "
                        + "VALUES (?, ?, ?, ?, ?," +
                        " ?, ?, ?, ?, ?);");
        statement.setString( parameterIndex: 1, game.getName());
        statement.setString( parameterIndex: 2, game.getProducer());
        statement.setString( parameterIndex: 3, game.getConsole());
        statement.setBoolean( parameterIndex: 4, game.isRented());
        statement.setInt( parameterIndex: 5, game.getDaysLeft());
        statement.setInt( parameterIndex: 6, game.getReviewCount());
        statement.setInt( parameterIndex: 7, game.getReviewSum());
        statement.setFloat( parameterIndex: 8, game.getReviewAverage());
        statement.setString( parameterIndex: 9, game.getEsrb());
        statement.setDate( parameterIndex: 10, Date.valueOf(game.getDateAdded()));
        statement.executeUpdate();
        statement.close();

        createdGame = readMaxId();
    }
    return createdGame;
}
```

The code above shows the way of adding a game to the database.First the program tries to create the connection to the database,using the PostgreSQL JDBC driver for Java. Then the program creates an SQL statement in Java in the form of a String.

The values in the statement are replaced with question marks which act as placeholders for the actual data. The placeholders are indexed, starting from 1, and replaced by the variables from the given **Game** object. After values have been assigned to placeholders the statement is executed and the connection is closed.

To get the game with an id from the database the method **readMaxId**, is used. What this method does is that it executes an SQL statement which reads out the object with the highest id from the game table. This can be done because the database is

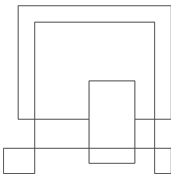


designed to grant ids serially with each added element having an id increased by one compared to the element added before it. Once the newest game has been found it is then passed back to the model using the return statement.

```
CREATE FUNCTION days_left()
    RETURNS TRIGGER
    LANGUAGE plpgsql
AS
$$
BEGIN
    IF PG_TRIGGER_DEPTH() <> 1 THEN
        RETURN NEW;
    END IF;

    UPDATE rentals
    SET days_left = rental_length_allowed - (CURRENT_DATE - rentals.date_rented)
    WHERE active = TRUE;

    RETURN NEW;
END;
$$;


CREATE TRIGGER days_refresh
    AFTER INSERT OR UPDATE
    ON rentals
    FOR EACH STATEMENT
EXECUTE PROCEDURE days_left();
```

To make sure that the data in the database is up to date, most notably information of how many days a game has left before it needs to be returned. To do this a trigger goes through the **rentals** table recalculating the value.

This trigger is run by a function r called **days_refresh**, this trigger itself is activated every time a new game is added or an existing game is rented. To prevent the infinite loop caused by a recursive call of days refresh, PG_TRIGGER_DEPTH is used to make sure the **days_left** trigger only runs once for each time **days_refresh** is activated.



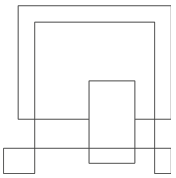
The Javafx library was used to create the GUI. To complement that, SceneBuilder was used to ease the design creation of FXML files, instead of manual code writing.

```
<TableView fx:id="table" maxHeight="-Infinity" maxWidth="-Infinity"
           minHeight="-Infinity" minWidth="-Infinity" prefHeight="262.0"
           prefWidth="427.0">
    <placeholder>
        <Label text="No games rented out!"/>
    </placeholder>
    <columns>
        <TableColumn fx:id="nameColumn" prefWidth="274.0" text="Game"/>
        <TableColumn fx:id="timeColumn" prefWidth="143.0"
                     text="Time left"/>
    </columns>
    <columnResizePolicy>
        <TableView fx:constant="CONSTRAINED_RESIZE_POLICY"/>
    </columnResizePolicy>
</TableView>
```

A commonly used Javafx element in the system are tables. Shown above is the code of implementation of a simple table in an FXML file.

```
public TableView<SimpleGameViewModel> table;
public TableColumn<SimpleGameViewModel, String> nameColumn;
public TableColumn<SimpleGameViewModel, Integer> timeColumn;
```

In the controller class for the FXML file which has the table, a declaration is needed to specify the data type each column will store. An adapter class is also needed to turn usual data types into their table safe counterparts. In the case shown above, **SimpleGameViewModel** is the adapter turning a String into a StringProperty needed by the table.

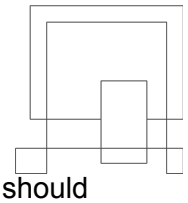


```
@Override
protected void init()
{
    viewModel = getViewModelFactory().getUserProfileViewModel();
    username.textProperty().bind(viewModel.getUsernameProperty());

    nameColumn.setCellValueFactory(
            cellData -> cellData.getValue().getNameProperty());
    timeColumn.setCellValueFactory(
            cellData -> cellData.getValue().getTimeProperty());
    error.textProperty().bind(viewModel.getErrorLabel());

    table.setItems(viewModel.getData());
    getViewModelFactory().getUserProfileViewModel().reset();

    reset();
}
```

Binding the data in the table with a variable of the adapter class needs to be done using the **setCellValueFactory** method and a lambda expression as shown above.

```
Platform.runLater(() ->
{
    try
    {
        model.modifyBalance( amount: 30, CurrentlyLoggedUser.getLoggedInUser());
        CurrentlyLoggedUser.updateInfoWithServer();
        reset();
    } catch (Exception e)
    {
        errorLabel.set(e.getMessage());
    }
});
```

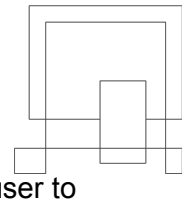
If a Javafx element needs to update without switching to another view, **Platform.runLater** must be used with a lambda expression. In the above shown code



Platform.runLater is used when the user adds money to their account which should change the number in the label of the GUI.

```
1 usage   MartyBobo +2
public void editUser()
{
    try
    {
        LocalDate dob = dobProperty.get();
        Period age = Period.between(dob, LocalDate.now());
        if (age.getYears() < 13)
        {
            dobProperty.set(null);
            throw new IllegalArgumentException(
                    "User has to be at least 13 years old!");
        }
        if (usernameProperty.get().length() < 5)
            throw new IllegalArgumentException(
                    "Username has to be at least 5 characters!");
        if (!emailProperty.get().contains("@"))
            throw new IllegalArgumentException("Email not in correct format!");

        User newUser = selectedUserProperty.get().getUser();
        newUser.setUsername(usernameProperty.get());
        newUser.setAddress(addressProperty.get());
        newUser.setName(nameProperty.get());
        newUser.setBday(dobProperty.get());
        newUser.setAdmin(selectedUserProperty.get().isIsAdmin());
        newUser.setEmail(emailProperty.get());
        newUser.setHasSubscription(
                selectedUserProperty.get().isHasSubscription());

        model.updateUserWithSQL(newUser);

        //change finished without error
    } catch (Exception e)
    {
        errorLabel.set(e.getMessage());
    }
}
```

The process of editing an already existing user starts from the UserEditViewModel.



In the client, while logged in as administrator, one can go to GUI and open a user to edit. All the properties will be loaded from the SelectedUser property. After clicking apply the program runs the checks on the fields. If any of the checks fails, the program immediately enters the catch block because of the errors thrown and appropriately updates the error label.

All of the instance variables are set except for the password because only the user can start a change for that, and the salt because it is generated automatically. The reason why the admin can't change the password is because it's encrypted using a PBKDF2 function (Locke & Gallagher, 2010).

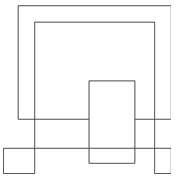
The model in this code refers to the RemoteModel, which is an interface that communicates with the server.

```
1 usage   chrfoyer
void updateUserWithSQL(User user) throws RemoteException, SQLException;
```

This is the method called within the Remote Model. The SQL in the method name refers to the fact that this method will communicate with the database using the driver.

```
Update the user info both in the user list and the database.
Params: user – The user to be updated
Throws: RemoteException – Thrown when an issue with the remote model occurs
        SQLException – Thrown when the connection to the database fails or the query is malformed
chrfoyer
@Override
public void updateUserWithSQL(User user) throws RemoteException, SQLException
{
    model.updateUserWithSQL(user);
}
```

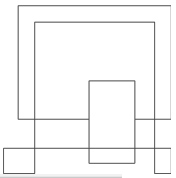
This is the implementation of the method on the server side. The model instance variable is the **Model** that exists on the server.



```
Syncs local user with database version
Params: user – is the user we want to sync
Throws: SQLException – in case of database errors
3 usages  Raedrim +1 *
@Override
public void updateUserWithSQL(User user) throws SQLException
{
    userDAO.update(user);
    refreshUserList();
}
```

This is the implementation in the model manager, that the method earlier shown calls.

The userDAO instance variable refers to the User Data Access Object that implements the interface responsible for communicating with the database using CRUD methods.



```
Updates an existing User with new information
Params: user – User with new data
Throws: SQLException – Thrown when the connection with the database cannot be established
3 usages  Raedrim +1
@Override
public void update(User user) throws SQLException
{
    try (Connection connection = getConnection())
    {
        Date date = null;
        if (user.getBday() != null)
        {
            date = Date.valueOf(user.getBday());
        }
        PreparedStatement statement = connection.prepareStatement(
                sql: "UPDATE users " + "SET password = ?, " + "email = ?, " + "name = ?, "
                        + "bday = ?, " + "has_subscription = ?, " + "balance = ?, " + "salt = ?"
                        + "WHERE username = ?;");
        statement.setString( parameterIndex: 1, user.getPassword());
        statement.setString( parameterIndex: 2, user.getEmail());
        statement.setString( parameterIndex: 3, user.getName());
        statement.setDate( parameterIndex: 4, date);
        statement.setBoolean( parameterIndex: 5, user.hasSubscription());
        statement.setInt( parameterIndex: 6, user.getBalance());
        statement.setString( parameterIndex: 7, user.getSalt());

        statement.setString( parameterIndex: 8, user.getUsername());

        statement.executeUpdate();
        statement.close();
    }
}
```

The implementation within the UserDao above is similar to the GameDAO in the way it executes updates.

First the program tries to create the connection to the database,using the PostgreSQL JDBC driver for Java. If the connection does not get established, or at any point of executing the statement the database throws an error, this method exits with a SQLException.

First a new Date value is created from the birthday of the user, because the Java implementation of the class LocalDate (which the program uses to store the value of the date of birth) is not compatible with the PostgreSQL implementation of the Date column.

The PreparedStatement is an unfinished SQL query where the question marks represent information that will be set using the information obtained from the viewmodel.

In each statement one of the question marks is set to their respective values from the user. Since the ViewModel already checked for any errors with the instance variables, this part runs without issues.

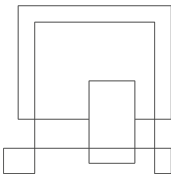
A ten argument constructor taking in every attribute

Params: `age` – Age in user
`username` – The username used to log-in
`password` – The password needed to log-in
`isAdmin` – Shows whether the user has administrator priveledges
`email` – The e-mail to contact the user if needed
`address` – The physical mailing address to use to send the games
`name` – The full name of the user
`bday` – The date of birth of the user
`hasSubscription` – Shows whether the user has an active subscription and can rent games
`balance` – The balance of money that the user has
`salt` – salt used for password encryption

Raedrim

Explanation of the values in the statement shown before. (See Appendices E-A -Client JavaDoc)

After setting all the missing values in the prepared statement, the SQL update is executed which the PostgreSQL database handles. The update command will use the given information, to find the existing user in the database and change the modified values.

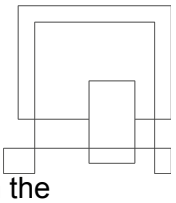


```
Syncs local user with database version
Params: user – is the user we want to sync
Throws: SQLException – in case of database errors
3 usages  Raedrim +1 *
@Override
public void updateUserWithSQL(User user) throws SQLException
{
    userDAO.update(user);
    refreshUserList();
}
```

After the statement is executed , and the connection is closed, the program returns to this method, where the refreshUserList method is called.

```
Syncs userList with database
Throws: SQLException – Thrown when issues occur with the database
6 usages  Raedrim
@Override
public void refreshUserList() throws SQLException
{
    UserList temp = new UserList();
    for (User user : userDAO.getAllUsers()
    )
    {
        temp.addUser(user);
    }
    users = temp;


}
```

This method takes care of syncing with the database so that the server, and the client always have up to date information.



The way this method works is that it creates a new UserList which is filled from the database, and then changes the UserList that the model uses to this updated one. Because the client uses the same UserList it will reflect the applied changes as well.

5 Test

The system was tested in numerous ways. Testing began with white box testing by using jUnit to look for any errors in the code. With knowledge of the implementation, the transparent testing method was used to test the most critical points of the system. The first tests ran were the basic jUnit tests. These tests were done on the basic model classes for gameslist and userlist and were based on the ZOMB+E method. An example of this is the getGame_M() method which tested whether it was possible to fetch multiple games.

As shown below, it passed the jUnit test because it was able to get several games and did not throw any exceptions.

```
@Test
void getGame_M()
{
    gameList.getGame(new Game( name: "Cod3",  producer: "Infinity Ward",  console: "PC",  esrb: "M"));
    gameList.getGame(new Game( name: "Cod4",  producer: "Infinity Ward",  console: "PC",  esrb: "M"));
    gameList.getGame(new Game( name: "Cod5",  producer: "Infinity Ward",  console: "PC",  esrb: "M"));

    assertDoesNotThrow(() -> gameList.getGame(new Game( name: "Cod2",  producer: "Infinity Ward",  console: "PC",  esrb: "M")));
}
```

```
✔ Tests passed: 1 of 1 test – 30 ms
C:\Java\jdk-11.0.2\bin\java.exe ...

Process finished with exit code 0
```

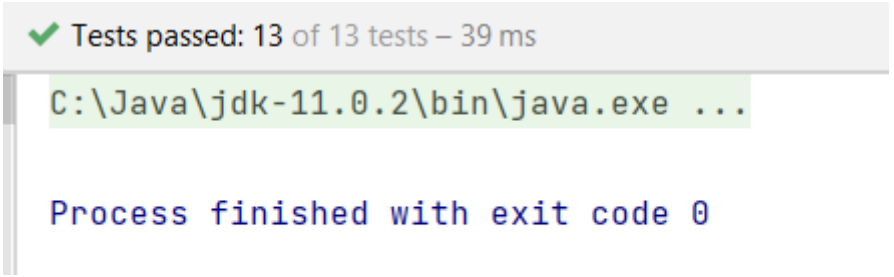
It was also able to pass every part of the ZOMB+E test method.

```
✔ Tests passed: 11 of 11 tests – 60 ms
C:\Java\jdk-11.0.2\bin\java.exe ...

Process finished with exit code 0
```

Another example is how the Userlist class was tested to see if everything was working as it should. As shown below, one of the important tests was the Z part of the ZOMB+E where a null user was added to the userlist, which passed the test because an IllegalArgumentException was thrown.

```
@Test
void addUser_Z()
{
    User user = null;
    assertThrows(IllegalArgumentException.class, () -> userList.addUser(user));
}
```

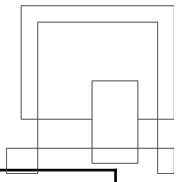
The ZOMB+E testing method also showed that the user list passed all tests.



In addition to the jUnit test which was unit testing for detailed design, system testing was also performed for the requirements analysis by following the V-model. This was done in the form of test cases for the use cases relating to the critical requirements. Below are several of the black-box tests in the form of test cases.

Use Case: Terminating a user
Test Case: Terminating a user while they have a game rented

Step	Input Field	Value	Expected result	Actual result
1.The administrator presses the manage users			Opens a list of users	Opens a list of users

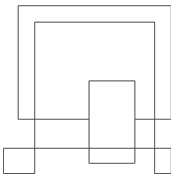


button				
2.The administrator selects the user to be removed by clicking on them		User clicked on	User is selected	User is selected
3.The administrator presses the remove button			Confirmation window opens	Confirmation window opens
4.They confirm their choice on the alert asking them if they are sure they want to remove that user			Exception is thrown	Exception is thrown

Use Case: Terminating a user

Test Case: Terminating a user while they have no games rented

Step	Input Field	Value	Expected result	Actual result
1.The administrator presses the manage users button			Opens a list of users	Opens a list of users
2.The administrator selects the user to be removed by clicking on them		User clicked on	User is selected	User is selected
3.The administrator presses the remove button			Confirmation window opens	Confirmation window opens
4.They confirm their choice on the alert asking them to remove that user			User is deleted	User is deleted



Use Case: Terminating a user

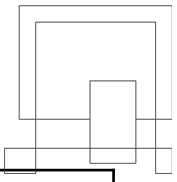
Test Case: Terminating a the user that is currently logged in

Step	Input Field	Value	Expected result	Actual result
1.The administrator presses the manage users button			Opens a list of users	Opens a list of users
2.The administrator selects the user to be removed by clicking on them		User clicked on	User is selected	User is selected
3.The administrator presses the remove button			Confirmation window opens	Confirmation window opens
4.They confirm their choice on the alert asking them if they are sure they want to remove that user			User is deleted, exception is thrown	User is deleted but stays logged in. No functionality. Every button throws a exception since user does not exist

Use Case: Terminating a user

Test Case: Terminating a admin

Step	Input Field	Value	Expected result	Actual result
1.The administrator			Opens a list of	Opens a list of

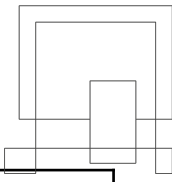


presses the manage users button			users	users
2.The administrator selects the user to be removed by clicking on them		User clicked on	User is selected	User is selected
3.The administrator presses the remove button			Confirmation window opens	Confirmation window opens
4.They confirm their choice on the alert asking them if they are sure they want to remove that user			Exception is thrown, user is not deleted	Exception is thrown, user is not deleted

Use Case: Renting a game

Test Case: Renting a game with renting privileges

Step	Input Field	Value	Expected result	Actual result
1.The renter browses the selection of games available for rent.				
2.The renter selects the game they wish to rent	Game clicked	Game clicked on	Game selected	Game selected
3.If the renter has renting privileges, they confirm they would like to rent the selected game.			Game is rented	Game is rented

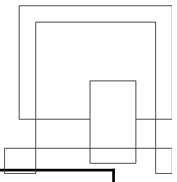


4.The game rental period begins on the game.			Rental period begins	Rental period begins
5.The game is added to the renter's list of rented games.			Game is added to list	Game is added to list
6.The game is shipped to the Renter's shipping address as specified in their profile.			Handled externally	Handed externally
7.The renter receives the physical copy of the game by mail.			Handled externally	Handled externally
8. The renter can play the game within the return period.			Handled externally	Handled externally

Use Case: Renting a game

Test Case: Renting a game without renting privileges

Step	Input Field	Value	Expected result	Actual result
1.The renter browses the selection of games available for rent.				
2.The renter selects the game they wish to rent	Game clicked	Game clicked on	Game selected	Game selected
3.If the renter has renting			Alert saying	Game is not

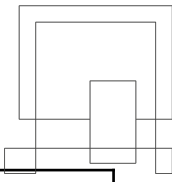


privileges, they confirm they would like to rent the selected game.			they are unable to buy a game without a subscription	rented, alert is thrown
4.The game rental period begins on the game.				
5.The game is added to the renter's list of rented games.				
6.The game is shipped to the Renter's shipping address as specified in their profile.				
7.The renter receives the physical copy of the game by mail.				
8. The renter can play the game within the return period.				

Use Case: Renting a game

Test Case: Renting a game without selecting a game

Step	Input Field	Value	Expected result	Actual result
1.The renter browses the selection of games available for rent.				

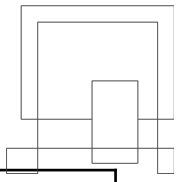


2.The renter selects the game they wish to rent	No game clicked	null		
3.If the renter has renting privileges, they confirm they would like to rent the selected game.			Exception is thrown, a game must be selected	No exception is thrown, rent button does nothing
4.The game rental period begins on the game.				
5.The game is added to the renter's list of rented games.				
6.The game is shipped to the Renter's shipping address as specified in their profile.				
7.The renter receives the physical copy of the game by mail.				
8. The renter can play the game within the return period.				

Use Case: Removing a game

Test Case: Removing a rented game

Step	Input Field	Value	Expected result	Actual result

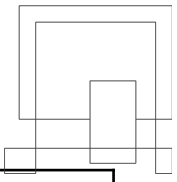


1.The Administrator browses the games in the inventory				
2.The Administrator selects the game they wish to remove		Game selected		
3.The Administrator is asked if they are sure they wish to remove the selected game				
4.The Administrator confirms the removal			Exception is thrown because the game is rented	Exception is thrown because the game is rented
5.The Administrator is taken back to the inventory screen				

Use Case: Removing a game

Test Case: Removing a game

Step	Input Field	Value	Expected result	Actual result
1.The Administrator browses the games in the inventory				
2.The Administrator selects the game they wish to remove		Game selected		

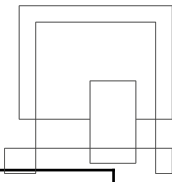


3.The Administrator is asked if they are sure they wish to remove the selected game				
4.The Administrator confirms the removal			Game is removed	Game is removed
5.The Administrator is taken back to the inventory screen			Back at inventory screen	Back at inventory screen

Use Case: Removing a game

Test Case: Removing a game without selecting game

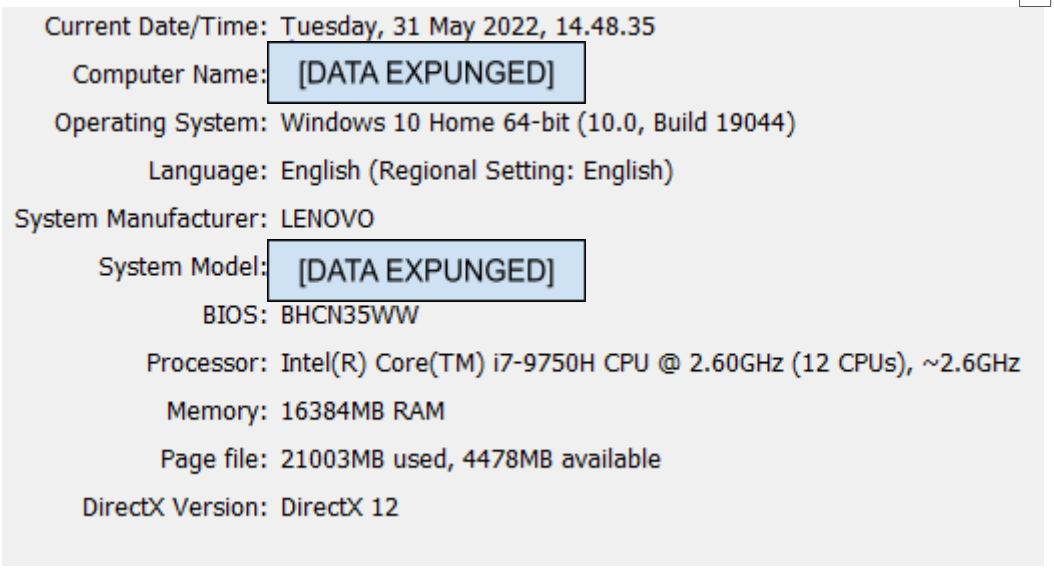
Step	Input Field	Value	Expected result	Actual result
1.The Administrator browses the games in the inventory				
2.The Administrator selects the game they wish to remove		Does not select a game	null	
3.The Administrator is asked if they are sure they wish to remove the selected game			Exception is thrown, no game selected	No exception is thrown, the return button does nothing
4.The Administrator confirms the removal				
5.The Administrator is				



taken back to the inventory screen				

Each step of the use cases was tested out in the form of test cases to see if it gave the expected result. As shown, the expected result almost always matched the actual result however there were a few exceptions. One of the exceptions was when an admin deleted a user that was currently logged in. The expected result was that an exception would be thrown, and while an exception was thrown, the user was still logged in to the client despite not existing on the server. While they were logged in, they had no functionality with the buttons because they did not currently exist. Another exception to the expected result is in the test case for renting a game without selecting a game. The expected result was that an exception would be thrown, however the actual result did not throw an exception. The rent button just did nothing when clicked. This was the same case for returning a game without selecting a game. It did not throw an exception as expected but instead the button just did nothing.

In addition to the test cases, the developers also tested to make sure that the requirement of opening the program within 60 seconds was met. There is a slight margin of error due to the manual operation of the stopwatch in the test and the results are also skewed because they were run on a computer with the following hardware specifications:



T1	T2	T3	T4	T5	T6	T7	Average Time
1.86 seconds	1.31 seconds	1.39 seconds	1.45 seconds	1.47 second	1.46 seconds	1.45 seconds	1.48 seconds

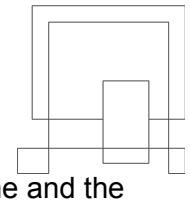
6 Results and Discussion

As seen in the test section, everything went as planned and the expected outcome almost always matched the actual result, with a few exceptions. By doing the acceptance testing, each requirement can be checked whether or not it was met.

Functional requirements

Critical priority:

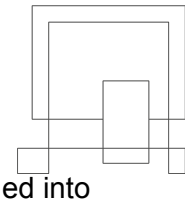
1. As a renter, I want to be able to rent games that I have access to for a specified amount of time, so that I can play them.



   a. This requirement is met because the user is able to rent a game and the rented game has a value of days left that shows how long they have before they must return the game.
2. As a renter, I want to be able to search through listings by different categories, ratings, or name so that I can more easily find a game that I want.
   a. This requirement is met in the game browser window it is possible to search for a game by the name, console, or rating.
3. As a renter, I want to be able to return rented games, so that I can rent new ones.
   a. This requirement is also met because the user can select a rented game and then press the return button to return it.

<u>High priority:</u>

4. As a renter, I want to login using a username and a password, so I can access the system on my own account on the system.
   a. This requirement is met because the user can enter their username and password to log into their own personal profile which is stored in the system, provided that they have previously signed up and created an account.
5. As a renter, I want to review my currently rented games, so that I can see if I can rent more games.
   a. This requirement is met because the user can log into their account and view all their currently rented games as well as the time left on those games before they have to be returned.
6. As a renter, I want to be able to register using my contact information including email, address, full name, date of birth, username, and password, so that I can have my own account on the system.
   a. This requirement is met because when the user clicks the signup button they are taken to a screen where they can enter their email, address, full name, date of birth, username, and password in order to create a new account with these values.
7. As an administrator I want to be able to add new items to the list so that the system displays up to date representation of the inventory.



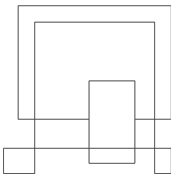
   a. This requirement is met because when an administrator is signed into the system they can add games to the inventory by entering the game's name, ESRB rating, and producer.
8. As an administrator I want to be able to change information about the existing items so that the system displays up to date information.
   a. This requirement is met because when an administrator is signed into the system they can select a game that they would like to edit and change its current information.
9. As an administrator I want to be able to remove games that are no longer in stock so that the system only shows items that are in stock.
   a. This requirement is met because when an administrator is logged in they can view a list of currently existing games and remove them as needed.

Medium priority:

10. As an administrator, I want to be able to edit the renter information so they match the real world information.
    a. This requirement is met because when an administrator is logged in they can select a user so that they can edit their information.
11. As an administrator, I want to be able to terminate renters in order to prevent and punish fraud.
    a. This requirement is met because when an administrator is logged in they can view a list of currently existing users and then select one. After selecting the desired user, the admin can then remove them from the system by pressing the remove button.

Low priority:

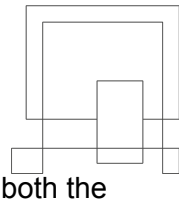
12. As a renter, I want to leave ratings for games I rented, to give feedback for other renters to view.
    a. This requirement is met because when a user is logged in and returns one of their games, they are prompted to leave a review with a value between 1 and 5.



13. As a renter, I want to pay for my subscription and for any arising late fees,so that I can use the platform.
    a. This requirement is met because when the user is logged in, they can add funds to their account as well as pay for their subscription.
14. As a renter, I want to be able to see the balance on my account, so I can make payments.
    a. This requirement is met because when logged in as a user, the user can click the payment button and view their current balance on their account.
15. As an administrator, I want to be able to issue a refund for an unsatisfied customer so that they want to continue using our services.
    a. This requirement is met because when the administrator is logged in, they can select a user and issue them a fund with the specified amount.
16. As an administrator, I want to be able to fine renters, so that I can discourage late returns.
    a. This requirement is met because once the administrator is logged in, they are able to input a value for how much they want to fine a selected user.
17. As an administrator, I want to revoke renting privileges of renters, so that only trusted renters have access to the system.
    a. This requirement is met because once the administrator is logged in, they are able to select a user and revoke their subscription which will prevent them from renting games.

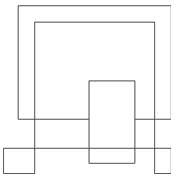
Non-functional requirements

18. The renter must be 13 years of age or older.
    a. This requirement is met because when a user signs up there is an age constraint when entering their birthday.
19. Users' passwords are not stored in clear text.
    a. This requirement is met because when relations in the database are examined, the passwords are encrypted using PBKDF2.
20. User guides must exist for users and administrators to explain core functions.



    a. This requirement is met because there exists user manuals for both the user and the administrator.

21. The database must be able to drop the schema and create the tables again from scratch.
    a. This requirement is met because the top code in the database DDL drops the schema and creates all the tables.
22. The client program must open within 60 seconds on a computer running Windows 10 and with a processor that has a clock speed of at least 1 GHz.
    a. This requirement is met because the program opens on average within 1.48 seconds.
23. The database must be implemented in PostgreSQL so that the current system administrators can use their current knowledge base to maintain it.
    a. This requirement is met because the database is created using PostgreSQL.



7 Conclusions

The primary goal of the system was to increase the value and decrease the cost of video games by renting out physical copies to consumers. It is clear that the decreased overall cost of rental increases accessibility for the mass market (Stegner, 2020).

However, the crucial point to consider is how the system designed and implemented works to facilitate that. The requirements set out at the start work to provide the needed actions to allow the end user to rent the game. Setting out the criteria by which the users can search through the catalog is important to receiving the best value from the system possible.

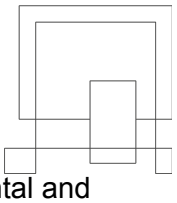
Additionally, the non functional requirements take account of the variety of interests the stakeholders have, which include legal and security considerations such as the user having to be at least thirteen years old to access the system .

For the renters to utilize their subscriptions to save money on purchasing games, they must be able to access a server containing the shared catalog of games to rent. For this reason a single user system was simply not enough, so a client server architecture was utilized. This also grants the stakeholders access and management to a finer detail when it comes to the inventory and user base.

A variety of design patterns make use of tested and proven ways to bridge individual pieces of code. In combination with the JavaFX, the model-view-view-model design pattern used a combination of binds and notifications to give the user a graphical user interface to help ease of use. Ease of use is key for the user to make sense of the information provided, which in the end makes for a better informed game selection process.

To improve efficiency and performance, the system was consistently revised and improved upon. On that front, the database is queried for a wide variety of information regarding the games and users.

By using the third form of normalization to design the relations, the query speed is increased because of the reduction in redundancies and multi valued attributes. The



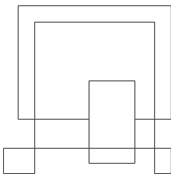
database is also responsible for calculating and updating the days left on a rental and the age of a user by using triggers.

To ensure that renters and administrators can effectively use the system, tests were used to drive the development. The results of the unit tests helped refine the classes, which were built upon.

From there, the test cases were developed to test how well the system tracked to the use cases made during the analysis. Finally, acceptance testing was used to see how well the system compared to the requirements set out by the stakeholders and agreed upon by the developers. The stakeholders added additional requirements they found to be key to generating business value effectively.

As shown by the results of the acceptance testing, the requirements were fulfilled. The administrators can add games and then the renters can rent them. Renters are able to browse and search the catalog of games available as well as see their current rentals. New renters can create profiles if they meet the criteria set out. Administrators even have a large suite of tools available to manage both their games and users, for example, changing user information, issuing fines, and paying refunds.

In summary, the implementation effectively leverages the design to meet the requirements set out during the analysis. On top of simply meeting the requirements on a binary level of yes or no, consistent quality of life updates by the developers have made the system easy to use. However, in a real world deployment of the system, there may be some useful additions to consider to improve the value generated for the stakeholders. These improvements will be elaborated upon within the project future.



8 Project future

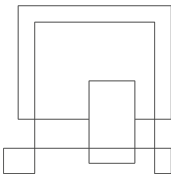
The system is currently not ready for launch. It has several missing features such as the fact that a physical game is never sent out to the user when he rents a game. Another issue is with the payment method because it is just a button that adds funds. This could be improved in the future by making the user put in their actual card information and paying with real money.

In the future the development team would like to add different types of subscriptions to the product. Currently the product only has one basic subscription for the users but in the future, the system could incorporate different subscription types that the users can choose from. These can for example give the user more time to rent games or increase the number of games that they can rent. One problem that the system currently has is that there is not a limit to how many games a user can rent so that is something that the developers would like to add in the future.

Another feature that the development team would like to add to this product is to expand it to new markets. Currently it is only available within Horsens but they would like to bring it to new markets nationwide. One thing that they would like to consider is to make region locks so that some games are only available within certain regions. An important feature that must also be added is the addition of new administrators. Currently the system only has one administrator which is not viable if the system is to expand nationwide since there will be thousands of users that need to be properly managed. The development team would also like to assign each game a barcode so that there would be better product tracking.

Another large adjustment that the development team would like to add is the availability of digital rentals. The current product works around shipping out physical copies of games but by adding digital copies it would make renting a game even more accessible for all and more appealing to a more modern audience.

Some last adjustments that the development team would like to add to the system is to fix some of the exceptions that were discovered in the test cases during acceptance testing where the expected result did not always match the actual result.



9 Sources of information

Davis, Z., 2022. *How Long To Beat.* [Online]
Available at: https://howlongtobeat.com/stats
[Accessed 26 February 2022].

Gilbert, B., 2020. *Insider.* [Online]
Available at:
https://www.businessinsider.com/video-game-price-increase-60-70-nba-2k-xbox-ps5-2020-7?r=US&IR=T
[Accessed 01 03 2022].

Kriz, W. C., 2020. *Gaming in the Time of Covid19,* New York: SAGE Journals.

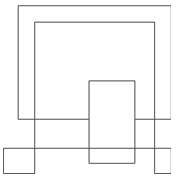
Locke, G. and Gallagher, P.D., 2010. Recommendation for password-based key derivation . [online] Computer Security Division Information Technology Laboratory .
Available at:
https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf
[Accessed 23 May 2022].

Rudis D. ir Poštić S. (2018) “Influence of Video Games on the Acquisition of The English Language", *Verbum*, 80, p. 112-128. doi: 10.15388/Verb.2017.8.11354

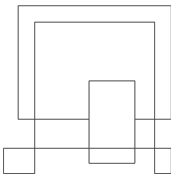
Stegner, B., 2020. *MakeUseOf.* [Online]
Available at:
https://www.makeuseof.com/tag/video-game-rentals-can-borrow-fun/
[Accessed 27 02 2022].

The PostgreSQL Global Development Group, 2021. PostgreSQL JDBC Documentation. [online] The postgresql JDBC interface. Available at:
https://jdbc.postgresql.org/documentation/head/index.html
[Accessed 20 May 2022].

Timofiychuk, S., 2020. Javadoc - intelliJ IDEA plugin. [online] JetBrains Marketplace.
Available at:



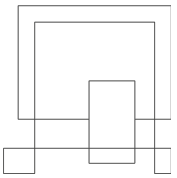
https://plugins.jetbrains.com/plugin/7157-javadoc
[Accessed 13 May 2022].



10 Appendices

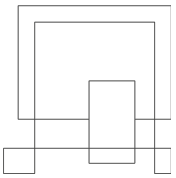
Source

A. Use Cases
    a. Use Case Descriptions.word
    b. Use Case Description.pdf
B. SCRUM Documentation
    a. Group 3 Burndown Charts.pdf
    b. Product Backlog.pdf
    c. Sprint backlog.pdf
    d. SCRUM Ceremonies Group 3.pdf
C. Class Diagrams
    a. Astah
        i. Activity_Diagrams.asta
        ii. Class_Diagrams.asta
    b. Exported Images
        i. ClassDiagramSummary.png
        ii. ClassDiagramSummary.svg
        iii. DatabaseAdapters.png
        iv. DatabaseAdapters.svg
        v. Login.png
        vi. Login.svg
        vii. Mediator(Client).png
        viii. Mediator(Client).svg
        ix. Mediator(Server).png
        x. Mediator(Server).png
        xi. Model(Client).png
        xii. Model(Client).svg
        xiii. Model(Server).png
        xiv. Model(Server).svg
        xv. RentGame.png
        xvi. RentGame.svg

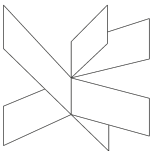


xvii. Signup.png
xviii. Signup.svg
xix. View.png
xx. View.svg
xxi. ViewModel.png
xxii. ViewModel.svg

D. Source Code
    a. Client Source
    b. Server Source
    c. Production ready Jar files
    d. Game Rental - Database DDL.sql
E. Source Documentation
    a. Client javadoc
    b. Server Javadoc
F. Guides
    a. User Guide - Admin.pdf
    b. User Manual.pdf
G. Database Design
    a. Source
        i. Entity_Relation.astah
        ii. Global_Relation_Diagram.drawio
        iii. Relational_Schema.word
    b. Exported Images
        i. Entity_Relation.png
        ii. Entity_Relation.svg
        iii. Global_Relation_Diagram.png
        iv. Global_Relation_Diagram.svg
        v. Relational_Schema.pdf
H. Project Description
    a. Project Description.pdf
I. Group Contract
    a. Group Contract_v2.docx
    b. Group Contract._v2.pdf



J. Other Diagrams
    a. Source
        i. Window Navigation.drawio
    b. Exported Images
        i. Window Navigation.dawio.pgn
        ii. Window Navigation.drawio.svg



Process Report: Game Rental

Physical media distribution in the modern age

Christian Foyer 315200

Martin Rosendahl 315201

Levente Szajkó 315249

Kruno Nerić 315258

Supervisor: Steffen Vissing Andersen

Software Technology Engineering

2Y

May 23th, 2022

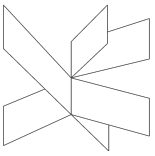
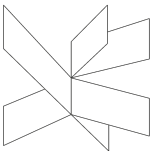


Table of content



1 Introduction

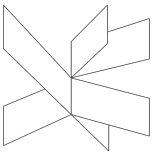
For the game rental project, we used a combination of SCRUM, Unified Process (UP), and agile development as the methodologies and frameworks to develop the system. However, even before we had the opportunity to learn about SCRUM and start the project period, we kept meeting minutes to document our process. We did start the Unified Process early during the very start of the inception. As soon as the project proposal was accepted by our supervisor, we were integrating the different disciplines in the inception as part of the Unified Process.

Once the project period began, the SCRUM artifacts became the main documentation of how the meetings went and the progress that was made. The ceremonies document contains notes about each sprint planning meeting, daily SCRUM meeting, sprint review, and sprint retrospective. The product backlog shows the user stories that define the system, which was broken down into smaller tasks during each sprint within the sprint backlog. For a graphical representation, we used the sprint burndown chart for each sprint that was updated during each daily SCRUM meeting.

2 Group Description

Our SEP 2 group consists of Martin Rosendahl, Christian Foyer, Levente Szajko, and Kruno Neric. Martin is half Danish and half Persian. He was born in Denmark but moved to the United States when he was young. He lived there until he turned 22 and decided to move back to Denmark to study software engineering at VIA University College. He grew up in California in the bay area. He has previous project experience from the SEP 1 group project. His previous experience in working with a group taught him how to effectively work with others and crucial skills in problem solving in order to avoid group conflicts. It also taught him to be more flexible and to be able to work around other people if needed.

Christian is a Danish American from the mountainous state of Colorado. After graduating high school, he worked several jobs including sommelier, goatherd, and pollster. However, it was not only the goats that he led. He has consistently taken



leadership roles in his work. Therefore, Christian has gained a lot of knowledge in the area of group working and time management, that he wisely used during the project work.

Levente is a Hungarian from Budapest, the capital. After graduating high school, he came to Denmark to continue his studies. Although not in software development, he had many student jobs where he picked up important skills about working in a team. He has previous project experience from SEP-1 which he used to add new ideas for the project.

Kruno is a Croatian from the city of Rijeka. He came to Denmark to continue his education after graduating from High School for electrotechnics and computer science. He participated in multiple programming competitions such as InfoCup and multiple regional coding competitions.

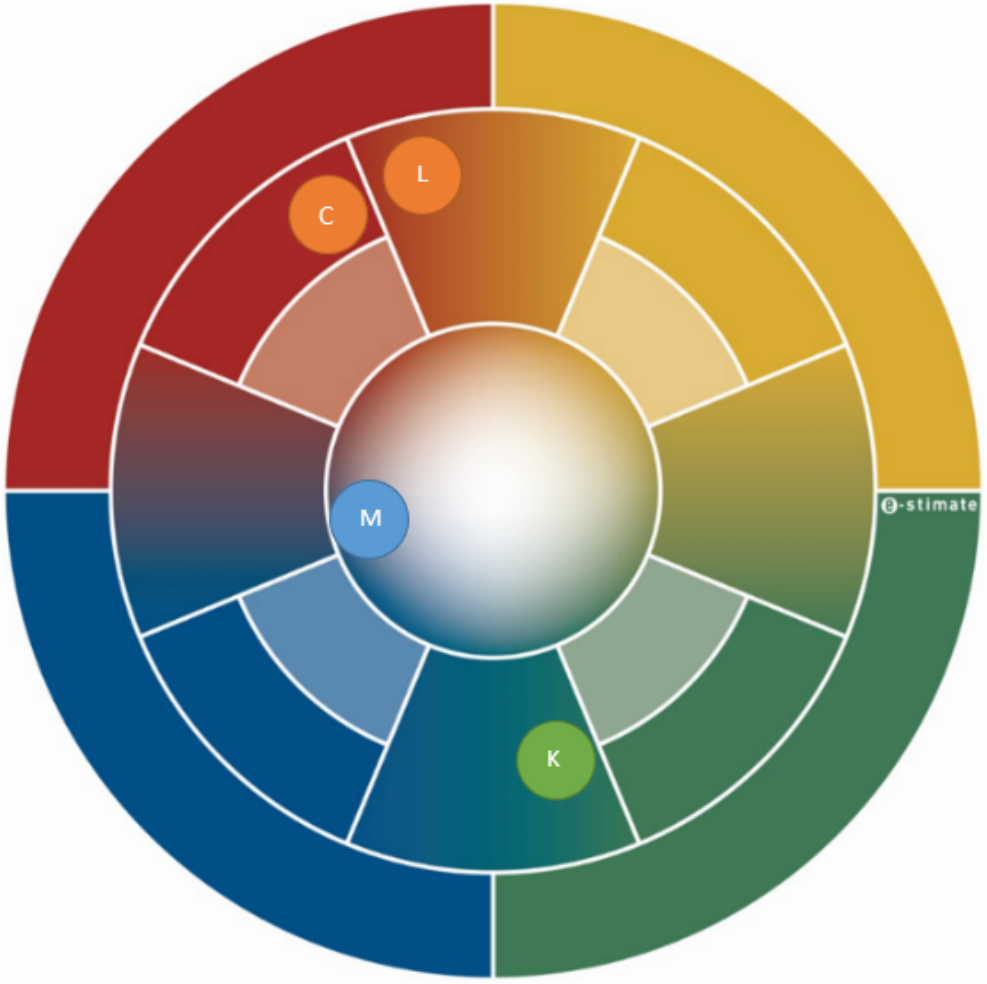
1. Communication										
Low context	1 Us	3	4	5	6	7	8	9	10	High context
2. Evaluating										
Direct feedback	1	2 Us	4	5	6	7	8	9	10	Indirect feedback
3. Persuading										
Principle first	1	2	3	4 Us	6	7	8	9	10	Application first
4. Leading										
Egalitarian	1	2 Us	3	4	5	6	7	8	9	10 Hierarchical
5. Deciding										
Consensual	1	2 Us	3	4	5	6	7	8	9	10 Top-down
6.Trusting										
Task-based	1	2	3 Us	4	5	6	7	8	9	10 Relationship-based
7. Disagreeing										
Confrontational	1	2 Us	4	5	6	7	8	9	10	Avoidant
8. Scheduling										
Linear time	1 Us	3	4	5	6	7	8	9	10	Flexible time

While our group had a good variety of cultural backgrounds, we were mostly unanimous with our placements on the cultural map. Everyone in the group preferred to be very low context and direct when in our communication.

When it came to group bonding and internal relations of the group, everyone preferred the egalitarian leading style while seeking consensus rather than competing. There was also no hierarchy established within the group, with no members whose say had more weight than others.

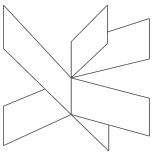
While we placed importance on bonding outside the project, we agreed that it was important to bond during actual work. When conflicts arose arguments rarely followed them, however conflicts were never ignored. All issues that came up were discussed calmly and without unnecessary escalation.

The biggest cultural conflict came with the scheduling because some members preferred more flexible meeting times and schedules. However when the issue was brought up it was swiftly resolved and we continued work without issue.



The picture above shows placement on the color wheel based on our personality profiles. Both Chris and Levente have red tendencies and were accordingly strong candidates for leadership roles within the SCRUM framework. With Chris' stronger focus on consensus and cohesion within the green space, he was chosen as the SCRUM master. Levente was chosen as the product owner because of his ability to stand by his decisions and demands.

Considering our different placements on the profiles, we tried to use the communication strategies offered. Kruno has strong green tendencies, so it was important that we



remained informal in our verbal communication to show a sense of togetherness. For Martin in the blue quadrant, we made sure to have our meetings and work flow structured to better hold attention.

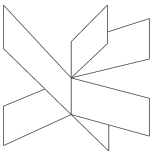
3 Project Initiation

After splitting up our respective groups at the start of the second semester, new groups started to form. The four of us got together as Group 3 of this semester, with Steffen as our supervisor. At this point we also wrote the first draft of our group contract.

In SEP 2 we each wanted to go bigger and better compared to the last semester. We had unique and interesting ideas to fit the “only” criteria: Client - Server application. We considered a dummy social media, and a simple game. In the end we choose to go with a blockbuster-like “library” system where you can rent out physical video games, as gaming is a common passion shared amongst the group members. At this point we also started to try and include the newly learned agile concepts from SWE, like SCRUM and UP.

Because none of us had earlier experience with working in such an environment, we had troubles in the beginning understanding of the key concepts. Because, in essence, sprints are timeboxed work windows, we agreed on having 4 workdays in a sprint. We also set up a preliminary sprint schedule where we agreed on where we would have the dates, and the exact time where the sprints would end. We talked about setting some deadlines where we would switch phases of UP. We discussed where inception would end, and elaboration would begin, however we didn’t set dates for these, as we didn’t have enough experience as a group to know when we would switch to the next phase.

We choose Chris as the SCRUM master because of his dedication to servant leadership and organization skills. Levente was selected as the product owner because he stands by his decisions, which best reflect the interest from the stakeholders.



This choice also made sense, because when we got our E-stimate the two members in the leading roles of the team had a plethora of red traits, which is important in these positions. The other group members became developers in the project.

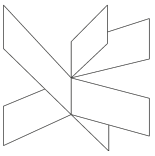
From our experiences from SEP-1 we brainstormed which ideas worked, and what we wanted to include in the system. After we had a general idea of what to include, we started working on the project description.

4 Project Description

The project description was the first document that we wrote as a group. It was easier this time around because we were able to reflect on our SEP 1 project description if we were unsure about how we should do something. We started off by getting the template for the project description from the VIA website. When we were in the description phase we had not learned SCRUM yet so we met up as a group and would look over the different sections and discuss what needs to be put where. We then assigned each other different parts of the template to fill out.

We would then start each of our group meetings by going over the different sections and seeing if there was anything that could be improved and decide what needed to be worked on more. We also had meetings with Steffen so that he could look over our project description and give us valuable feedback about what worked and what did not work.

The most challenging part of the project description was connecting the background description to the problem statement without excluding or introducing new information. We really struggled with this because we were not sure how to format our problem statement so that it would match the background description. We had to take several different drafts of the problem statement until we found one that made sense. We also struggled a little with finding sources and properly incorporating them into our text. We had to refer back to our SEP 1 project description to see how we did it last time.



5 Project Execution

In the project execution phase, we followed the SCRUM method. We had 5 sprints where each sprint consisted of 4 days. We would start each sprint with a sprint planning meeting where we would decide the sprint goal of what we wanted to accomplish. These goals were broken up into tasks that we wrote in the sprint backlog and then self organized to decide the responsible party. We would then spend the sprint working on our tasks and if we completed them early we would assist someone else with their tasks.

At the end of each sprint day we would hold a daily SCRUM meeting which was led by Chris since he was the SCRUM master. In this meeting we would discuss what tasks had been completed and what tasks we would like to work on for the next sprint day. We also talked about any problems that we experienced during the sprint day. We found that we much rather follow the SCRUM method than the waterfall method because it was really helpful with setting up goals and assigning tasks. The thing that we really enjoyed about the SCRUM method is that it is very adaptable whereas the waterfall method is not.

At the end of our 4 day sprints we would hold our final daily SCRUM, followed by the sprint review which was led by Levente who was our product owner, and then ended with the sprint retrospective which was once again led by Chris. In the sprint review we would go through the product backlog and do a demo so that our product owner could have a better understanding of what was completed and to see if our work had met his definition of done. In the sprint retrospective meeting we would talk about the things we should start doing, stop doing, and continue doing.

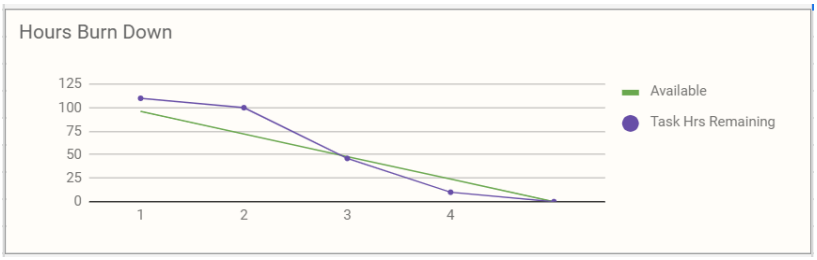
In addition to following the SCRUM method, we also followed the tenets of Unified Process. We found that SCRUM and Unified Process go really well together because in SCRUM you do things step by step and that lined up with the different phases really well. One of the key aspects that both SCRUM and Unified Process have in common is iteration. In iterative development, we work incrementally in time boxed cross-discipline iterations.

We followed the four phases of Unified Process, starting with inception. In inception we had to do the planning part of the project. This is when we came up with different ideas for what we could make our SEP project on. The next phase was elaboration and during this phase we created the problem statement and worked on our project description. After elaboration we had construction and this was the most difficult phase for us. This is where we began wrapping up our code and doing some of the documentation. In the transition phase, which is the phase we are currently in, we finished the testing and documentation of the final version of the system. It is an important part because this is where we prepared the system and documentation for deployment.

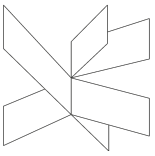
1	PB ID	Task Title	Comments	Responsible	Estimate
20	4	ER diagram first draft	See domain model	Chris	5
21	4	Global relation first draft		Kruno	5
22	4	Relational schema		Kruno	3
23	4	Creation of the tables		Chris	8
24	4	Constrainsts		Chris	10
25	4	Queries to test		Lev, Martin	13
26	18	Balance user side	Payment and adding balanc	Lev, Martin	5
27	14	Refund		Lev, Martin	5
28	16	Fine		Lev, Martin	6
29	17	Revoke user priveledge		Lev, Martin	9
30	12	Ratings		Lev, Martin	13
31	12	Terminate user		Lev, Martin	5
32	6	Check for date using triggers	All triggers	Chris	5
33	14	Transactions table	Create transactionDAO and	Kruno	5
34	4	Add game to a database table		Chris, Kruno	8
35	6	Retrieve the games rented by user		Chris, Kruno	5

For an example of a single sprint from end to end, sprint 3 was a very ambitious sprint where we refined the estimates of our hours for each task. During the sprint planning meeting, we looked at the items from the product backlog that we needed to address. From there, we broke the tasks down into smaller subtasks and then self organized for who would tackle each item.

After discussing what was completed at the daily SCRUM meeting, we marked what was completed on the sprint backlog. Correspondingly, we added up the remaining hours of tasks for the burndown chart. This provided a sense of visible progress as well as a meaningful way to track velocity to see if we were on track.



On sprint 3 in particular, we had 110 hours of tasks scheduled on the backlog, whereas we had 96 work hours of group work. After the second day, we were



back on track and we ended up completing all of the tasks by the end of the sprint. During the retrospective and review, we reflected on the work completed and the state of the system at the time.

6 Personal Reflections

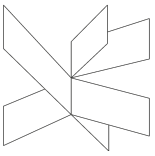
Martin:

I had a great time this semester with my current SEP group. It was a different experience from my last semester group because there was a lot more comradery but we still managed to complete our tasks and everyone did their parts.

The content of our group contract was just expectations about what we expect from the other group members. We focused on being on time and not having any unexcused absences from both the group meetings and from class. This is something that I think that our group did really well. There were a few exceptions but for the most part everyone was on time and if anyone was late or were unable to show up, they checked in with the group to let the rest of us know first and they always made up the missed work on their own time.

This SEP project was a lot more complicated than the first SEP project and involved a lot more work to complete it. We did this by assigning tasks and dividing up the work and if one of us finished early we would assist a group member with their work so that everyone could finish their work on time during the sprint. I feel responsible for this group project because the game rental was an idea that I came up with when our group first started brainstorming ideas for our SEP project. There were also many classes and methods that I worked on that were crucial for the project. An example of this is a bug we had in our removeUser method where when we removed a user it would remove them from the GUI but it did not delete them from the database.

Some examples of how the group contract has had a direct impact on the success of our group is how everyone came to our group meetings on time and we never had to issue out any of the consequences. We also followed the group contracts



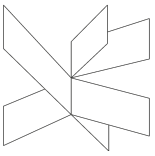
communication policy really well by messaging each other in discord by creating different channels for room-booking, code, important, and general. This was very useful in keeping track of all our group information in an efficient way and making it easy to share things with each other.

Some adjustments I would make to the group contract would be to add some informal meetings. In our last SEP group we had a lot of informal meetings which were a great way to destress in addition to helping us get to know each other better and I think that it could have been nice to have it for our current group as well.

My group worked really well together. Everyone was willing to ask for help if needed or to offer it if they could see someone was struggling. We followed the SCRUM method for our SEP project so everyone was assigned tasks and if they finished their task early they would help a group member with their task so that we could always meet our deadlines. It was not uncommon for us to work in groups of two when tackling a problem.

I believe that everyone delivered to the maximum because everyone always completed the tasks they were assigned, even if they had to use their time outside of group meetings to do so. We were also good at utilizing everyone's expertise, for example with Kruno who was exceptional at Scenebuilder and handling anything view related or Chris with his knowledge of databases. Levente performed really well, not just in the coding but also in the SCRUM aspect where he was the product owner and had a very clear idea of what he wanted and how to relay it to the rest of the group. The way we first formed the group also reflected our E-stimate profiles really well, where Chris and Levente were mostly red, they ended up becoming the SCRUM master and product owner. In those roles they both got to utilize their leadership qualities whereas me and Kruno who were mostly blue really excelled when given specific tasks that we could work on with specific directions.

I would say our motivation for this group was pretty consistent throughout the project. I think a point where we lost the most motivation was during sprint 3 for day 1 and day 2 because we did not manage to finish many tasks and were above the burndown chart line. This changed pretty quickly by day 3 where we caught up and then everything



went well after that. There were also some other issues when we encountered bugs or had trouble with Github when merging commits but we were always able to resolve them. I think what motivated our group the most was that we enjoyed the project we were working on and we had a lot of fun working together.

Having a multicultural group was exciting. For this SEP group we had Kruno who was from Croatia and Levente who was from Hungary. This was my first time meeting people from those countries and it was always fascinating to hear about where they came from or what life is like there. I do not think being multicultural had too much of an impact on the group work itself but it was definitely an insightful experience that taught me something new.

Since I had a new group this semester I had to learn how to work together with Levente and Kruno. It was a little challenging at first since their personalities were pretty different from my group mates in the last SEP project but as I got to know them better it became a lot easier. I think working with Levente was the hardest because he wasn't always the best at explaining things when he disagreed but throughout our time together he got much better at it. I have gotten used to the way Levente works and am able to work around it. Learning about our E-stimate-profiles helped a lot because Levente was mostly red and it made it easier to understand the way he works and thinks.

Throughout our project we kept in contact with Steffen. We met with Steffen around once a week so that he could look over what we had done and answer any questions. I am very satisfied with having Steffen as a supervisor because he was always happy to help and he was always able to answer any questions we had or give advice about how to approach things.

Levente:

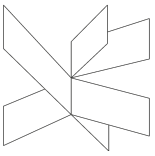
Our group formed on the remnants of the two disbanded groups after the events of SEP-1. Eventually, Kruno and I joined up with the 2-person team Chris, and Martin. Using our reflections of SEP-1, we laid the ground rules for the group contract, which were in effect until the end of the project period.

In the group contract we set up the rules for the approved number of absences, and the number of working hours we would put into the project. In the end, after some minor adjustments, we followed up on a group contract quite well. The only difference I would include for the next semester is to include stricter rules on the exact work hours.

After brainstorming ideas, we decided to stick with the Blockbuster-like rental project. When we chose the project game rental, I was quite enthusiastic about it since it is one of my favorite mediums. I think the analysis section of our Project Description was thorough because we could incorporate relevant newspapers, and articles from scientific journals. The problem domain encapsulates the ever-increasing cost of new games, which may be expensive for end-users who would be interested in a rental-based approach.

Our project was to create a system for game rentals in Horsens. This was a realistic goal, that with enough work put into it, would result in a working solution. We also decided, from a technical standpoint, that we would use RMI and a database in our final solution. For collaboration in the shared documents, we used Google Documents, and for drawing out early versions of the class diagrams we used Astah.

During the early days of the SEP classes when they were not in the project period, we had trouble using the SCRUM methodology to its full potential, as none of us had previous experience working in such a workflow. The group also assigned me as the



Product owner, so it meant that I had even more expectations put upon me, which I made sure to meet with external research, and information to be the best in my assigned role.

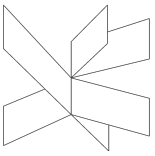
I felt quite responsible for the group project because when I got the role of product owner every other group member had high expectations from me. I always made sure to give accurate use cases that we could easily map to real-world scenarios and make sure after we completed tasks from the sprint backlog, we would have a correct order of priorities. I also tried to uphold a strict definition of don when refining the product backlog. This made sure that the stakeholders' interests were always accounted for, which fulfilled my role as the product owner.

Thankfully, after SWE classes and the supervisor's help, we began to fully understand the key concepts of SCRUM, which we used effectively during the project period. We took extra steps to follow the disciplines of UP to make sure we didn’t accidentally do a project that resembles Waterfall. By the end of the project period, I can now say that I see the merits of working in SCRUM, and for the next semester in SEP-3 I am sure we will use it to its full potential.

I also made sure to look up new practices, and tricks to make working in our IDE more efficient, and to make sure our code falls to the current industry standards. In the implementation, I took care of most of the ViewModels, and most of the basic methods in the Remote Model, to make sure our client communicates with the server using the RMI protocol.

I don’t think our different cultural backgrounds introduced any annoyances or problems into our workflow as a group. Through the course of the semester, I feel as though all of us opened up to each other's cultures, by sharing our movies, sweets, and other foods.

During group work, we all figured out quite early on the field that we wanted to contribute to the most. Using that, we managed to assign work for everyone that was challenging but insightful. In the end, we cannot say who worked “most” or who put in



the “most” amount of work, as everyone contributed their share to make sure the project was completed.

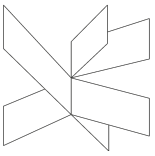
I was always motivated to work together with this group because even though we aimed to uphold a professional work environment, we always made sure that we all had fun fooling around and making jokes all the time. I gave my fullest to make sure I contributed what I could to the project, through which I managed to deepen my understanding of the practices from SDJ-2. For SEP-3 I want to continue working in a similar setting, because it was both challenging and informative, which made developing new skills the easiest.

During the project period, we made sure to contact our supervisor, Steffen, when we had serious issues with the code itself, or questions about our deliverables. I think these short sessions with our supervisor were extremely insightful and helped us get through the difficult parts of the project.

Overall, I think we made a project that fulfills all the criteria given this semester by utilizing the PBL-based nature of SEP to its fullest potential and even grew closer as friends during the completion of the project.

Kruno:

This SEP project was quite a different experience from what I experienced last semester. In the group I still had Levente from my last semester group and Chris and Martin from an arguably more successful group. The workflow was different from last semester due to us getting vastly different instructions on how to work on our project.



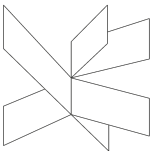
At the beginning, when we were first introduced to the concept of SCRUM and Up way of work, we had a few difficulties. However we quickly got used to how to work with it and it helped us a lot to not get overwhelmed with the amount of work that needed to be done. It was definitely easier to work then when we were using the waterfall principle because we didn't have to aim for a final product right from the start of the project. We were able to efficiently divide the system into smaller parts which allowed us to focus on each part and not spread our attention too thinly. Each member fulfilled their respective duties. A good example of this would be when our product owner, Levente, decided that a task on our product backlog should be moved to a higher priority since it had more importance for the system.

We encountered almost no significant issues when it came to our group work coordinatization. The only thing that we took notice of in the beginning of the project is that we need to be stricter with enforcing the set times either for meetings or breaks. Our group contract was often referred to whenever one of us was unsure of the exact rule we stated. Because we did a good job of stating the initial group contract there was no need to do many or any changes.

The problem-based learning which our course favors is the best way for us to develop the technical skills we will need in the future and the group work that was enforced allowed us to learn from each other topics and subjects that we were less experienced or knowledgeable about. I personally enjoyed the style of designating roles within our group according to the SCRUM principle because it allowed us to always know who in the group to address in case we had any issues or needed clarification on something.

I am also satisfied with my personal role within the group because it fit in well with my E-stimate personal profile. With my primary color being green I always sought to make sure all group members were on the same page and that there was no miscommunication within the group.

When it came to work I enjoyed when I worked with someone else on a part of the code or documentation. Because of this I often paired up with Levente when Java coding, or Chris when it came designing and implementing GUI.



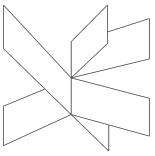
In summury I am very content with our groups' work as well as dynamic both in and outside the scope of project work.

Chris:

It is always a pleasure to learn new strategies at the same time you have the opportunity to implement them. That was the case this semester, as we learned the theory behind software engineering in SWE while we worked on the early stages of our semester project. The Unified Process was a nice departure from the very linear Waterfall methodology. The two largest aspects that I see as improvements are integrating disciplines and the iterative approach.

These two aspects together helped the group through our inception and elaboration phases. We were able to make our user stories short and only relating to the most key aspects we needed. From there we were able to diagram a very small initial iteration of the core of renting a game within a single user console command. From a theory standpoint, we were starting our iterations small so that we could integrate the disciplines as the system grew. From an interpersonal standpoint, these small iterations proved to the group that it was something that we could do. We enjoyed the iterative approach because it gave us early visible success and helped to mitigate the highest risks early in the project.

Another key project framework that we learned about and utilized was SCRUM. Early on, before we had learned about the roles available, I expressed an interest in a leadership position because of my experience and will to improve within my studies. When I learned about the SCRUM master position, I became keenly interested because of the focus on servant leadership and assisting the team to reach our shared goals.

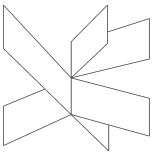


Similar to my group mates, I feel that the positions available as well as the overall framework matched our individual and cultural profiles well. Both Levente and I have red traits and took both of the leadership roles within the SCRUM framework. Martin and Kruno worked as the developers and used their blue and green traits to help in active listening and group cohesion.

In terms of culture and project based learning, our composite cultural profile suited the framework well. A core part of the SCRUM framework is the self organizing teams. Using the 8 dimensions defined by Erin Meyer, we all agreed on having an egalitarian leadership structure and using consensus based decisions. Although our individual cultures leaned on that side of the axis for both of the dimensions, we all individually learned further left from our respective cultures. With the flat hierarchy and consensus based decision process, my role as SCRUM master really worked to assist and facilitate the work being done.

We reflected deeply as a group after the final sprint to see how we felt about the framework. We had taken the recommendation from our supervisor to start our sprint days around noon to allow for greater flexibility. However, we felt that the days of our sprint should have started at the beginning of the work day to make each of the days more discrete and provide for more concrete reflection. Within my role as a SCRUM master, this would make the daily SCRUM meetings a more meaningful small milestone, instead of an after lunch meeting.

This was different from my previous work related leadership experience, because most organizational structures I had worked with had a very top-down focus. In particular, my work in hospitality rarely used self organizing teams and decisions frequently followed the precise chain of corporate command. Additionally, in my early jobs before I had the chance to lead, many of my managers used a theory X approach within McGregor theory. By this, I mean that they took an authoritative approach which assumed that workers did not have an interest in being there and therefore made strict deadlines. My personal approach to leadership, however, is more theory Y, wherein I focus more on the process and try to foster initiative within the group.



Finally, my role as the SCRUM master also gave me a chance to work with the Herzberg theory of hygiene and motivation factors. Each daily SCRUM meeting gave us a chance to address any issues with workflow. Then, at the end of the sprint, we could speak more generally and gather suggestions to better how we worked during the sprint retrospective. These discussions gave me a chance to fix some hygiene factors like the work conditions within the study rooms and frequency of breaks. When it comes to the motivation factors, the ability of group members to choose the tasks they want is very important for fostering a sense of responsibility.

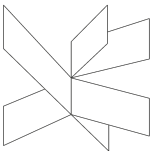
The new theory we learned was a great boon to our development process, while the theory we had learned in the previous semester helped us understand management styles and motivation.

7 Supervision

Our group was very satisfied with the supervision we had received. For the project we had Steffen as our supervisor. We met with him around once a week and he was always of great help. He always made time for us and was always happy to help. He was always there if we were unsure of something or had any questions. He was also really good at guiding us in the right direction if we were unsure how to tackle something or which direction to take the project in.

8 Conclusions

Overall, we all had a great time working together on this SEP project. We liked that we all had the same sense of humor and could always fool around while still getting work done. We believe that the SCRUM method was very helpful due to its rigid structure where we always knew what needed to be done and when it was due. We often found ourselves pairing into groups of two to tackle more complicated problems and always completed our tasks. While working together at the start was a little difficult since we had not gotten to know each other yet or our work styles, over time it got a lot easier.



Appendices

See the appendices in the Project Report