**1. A problem definition that describes this entire e-services type of system and details**

**specifically what problems the Xtra-Vision software system will solve.**

Xtra-Vision software system is a video rental kiosk which allows customers to rent any movies from Xtra-vision express machines located in Ireland. Customers can return the movies either to the same machine or another one in the network. Each disc has an electronic tag, which is used to scan them, and thus to track all the information about the rentals. There are three different categories for movies: coming soon, latest or new release and not new.

Firstly, The working prototype must allow customers to rent a movie using only a debit/creadit card. In case, the customers wants to get the receipts and details about new movies releases, they must provide the email address. In the first transaction, customers can get one disc for free using the code FREE123, and the first transaction is limited to two discs. Then, up to four movies can be rented at one time using the machines. There are special offers when we rent DVD & Blu-rays, for example:

1. Unlimited movies service: No more late fees, we can rent two movies at any time and rent them as many times as we like, for 8.99 euros. To rent more movies, they must bring back the two movies rented

2. Rent one movie for 2.99 euros for 1 night

3. Rent two movies for 2.99 euros each for 2 nights

4. Rent three movies for 2.99 euros each for 3 nights

Secondly, the cost rental is charged immediately after the customer rents a movie, but if there is an additional because of late disc returns the customer must pay the delay when he/she returns the movie. On the Xtra-vision website, we can see the following charge penalty fees:

“Will I be charged if I am late returning a disc?

If you want to hold onto your disc for longer there is no problem. For each additional day you keep it past 8pm, you’ll be charged €1.50. If you hold onto your disc for the maximum rental period – 10 days, you will be charged a maximum fee of €15 and the disc is yours to keep.”

Thirdly, as I have already said Xtra-vision Xpress vending machines is certified as a PCI DSS Level 1 Service Provider, the highest level of assessment available. This system is listed on both the Visa Global Registry of Service Providers and the Mastercard Compliant Service Provider List.

Fourthly, X-tra-Vision software will solve the following problems:

1. Customers can rent a movie on their own. Not employees from the company are required to be present at that moment.

2. Xtra-Vission company does not need to rent/own a storefront where customers can rent movies

3. There are many Xpress machine locations where customers can rent/return discs. This gives more flexibility for the customers

4. The system allows customers to rent a disc quickly, only with a debit/credit card. We do not need to log into the system (no membership is required)

5. Same system could be used to rent other products.

Fifthly, objectives of the new system:

1. To provide a rapid rental process

2. Simplify the rental process

3. To improve customer satisfaction

4. To promote our business, we can use direct email to advertise the new releases movies and thus to attract more customers

And scope the new system:

1. Rent procedures

2. Return procedures

3. Allow customers to view machine locations

4. Allow customers to open and closed unlimited movies account

5. Allow customers to view the movie catalogue

6. Provide special offers to customers

Lastly, as a drawback, I can say:

1. Machines need maintenance

2. The disc technology provided to the customer is older than other alternatives such as Netflix.

**2. A full list of requirements regarding what the system.**

The list below is a short set of requirements for Xtray-vision software system collected after reading the assignment and the company webpage. We also discuss the requirements between us.

R1. Work out how much the customer owes for late rental returning

R2. System must provide special offers to customers

R3. Print receipts is optional, it depends on every customer. Customer must provide an email address

R4. We must track all the information, including when discs was rented, returned and sold

R5. Keep a record of all the machine locations (Eircode, Address, Town, County, etc.)

R6. We must allow customers to open and close an unlimited account to rent movies

R7. Customers needs to see all the movies to rent in one machine. Keep a complete list of all the movies and their details (title, description, link video, type, genre, etc.)

R8. System must allow customers to pay using a debit/credit card.

R9. Customers use cards as login credentials.

R10. Record details of a rental transaction (start date, return date, number of days, etc.)

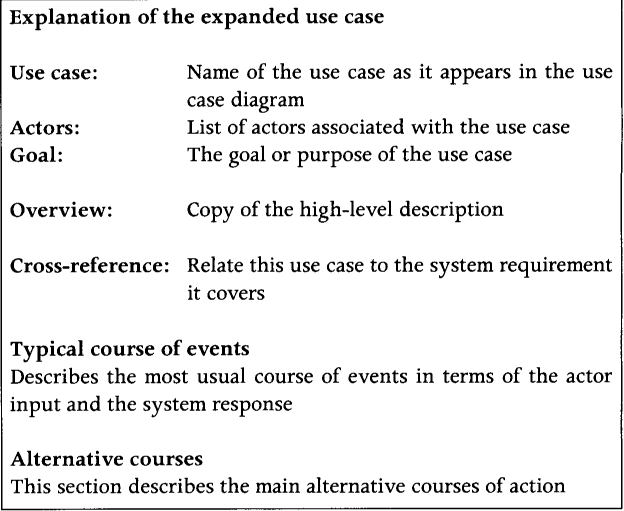
**3. Four Use Case diagrams that fully describe four different scenarios in which a user**

**will use the Xtra-Vision movie kiosk. Include at least one ‘failure’ scenario along with**

**the traditional ‘successful’ scenarios.**

I created an extending use case called checkout, which adds additional behaviour to the return movies use case. It could have been illustrated in the alternative paths, but as the extra functionality has many steps I preferred to model it using another use case.

I used the standard provided by the book for the use case descriptions. The image below show the template.

****

**Use case descriptions:**

Id: 1

Use Case: **View Locations Machine**

Precondition: The locations list is up to date

Actor: Customer (User)

Goal: Customer wants to know where the X-press rental machines are located to rent/return a disc

Overview: Customer starts to operate the machine and then (s)he views all the locations where machines are. Customer also can see details about every location

Cross-reference: R5

**Typical course of events:**

Actor action System response

1. The customer clicks on Locations button

2. Displays all the locations where the machines are

3. Customer types values to search for a specific location to see

its details (Address, Eircode, Town, County, etc.)

4. Use case ends

Priority: High

---------------------------------------------------------------------------------------------------------------------

id: 2

Use Case: **Open Unlimited Movies account**

Actor: Customer (User)

Goal: Customer wants to get an Unlimited movies account to rent as many times as (s)he likes

Overview: Customer starts to operate the machine, then starts the Unlimited movies service

Cross-reference: R2, R3, R6, R8, R9

**Typical course of events:**

Actor action System response

1. The customer taps on Unlimited Movie button

2. Displays the options to start/cancel/rent the Unlimited Movies service

3. Customer starts the service clicking on the

Start your Unlimited Movies button

4. Customer clicks on checkout button to see details about this service

5. Displays details about this service (total cost, etc.)

6. Customer taps on Checkout button to proceed to payment

7. Asks the customer to insert the card all the way into the reader

8. The Customer inserts the debit/credit card into the

chip and pin reader

9. Asks the customer to insert her/his pin number

10. Customer inserts the pin number

11. Verifies card and pin number

12. Processes the payment (8.99 euros), creates a customer account, and sets the monthly subscription on customer account

13. Customer removes the card from the reader

14. (Extension Point: **Send receipt by email (id 5)**)

15. Use case ends

**Alternative courses**

Step 9. Card can not be read. Customer removes the card from the reader. The use case resumes at main ‘success’ scenario step 15

Step 11. Card is not enabled to use by the bank. Customer removes the card from the reader. Use case resumes at main ‘success’ scenario step 15

Step 11. Pin entered is incorrect. Customer removes the card from the reader. Use case resumes at main ‘success’ scenario step 15

Step 12. Payment processing fails, use case resumes at main ‘success’ scenario step 15

Step 12. Customer already exists in database, so system processes the payment, gets customer details from database and changes her/his account to unlimited movies. Use case resumes at main ‘success’ scenario step 13

Step 12. Customer already has an unlimited movies account. Use case resumes at main ‘success’ scenario step 15

Step 14. **Send receipt by email** fails, use case resumes at main ‘success’ scenario step 15.

Priority: High

---------------------------------------------------------------------------------------------------------------------

id: 3

Use Case: **Rent movies**

Actor: Customer (User)

Goal: Customer wants to rent movie(s)

Overview: Customer starts to operate the machine, then (s)he chooses movie(s) from the list displayed by the computer. Finally, customer pays, and then leaves with the disc(s).

Cross-reference: R2, R3, R4, R7, R8, R9, R10

**Typical course of events:**

Actor action System response

1. The customer taps on Rent button

2. Initiate **View Movies (id 4)**

3. Customer chooses (a) movie(s) to rent

4. Verifies there are/is a disc(s) for the movies selected

5. Display the movies selected with the most important info (title, genre, availability, etc.)

6. Customer clicks on checkout button to see details about the rental

7. Shows details about the rental , for example how many movies were selected to rent, cost per disc, total cost, rent date, return date, number of

days customer will have the disc(s), etc.

8. Customer taps on Checkout button to proceed to payment

9. Asks customer to insert the card all the way into the reader

10. Customer inserts the debit/credit card into the

chip and pin reader

11. Asks customer to insert her/his pin number

12. Customer inserts the pin number

13. Verifies card and pin number

14. Gets the customer details from database and checks customer is allowed to rent the movies (the selected quantity is allowed). Then, processes the payment with the total amount. Then rental details are saved in the database

15. Customer removes the card from the reader

16. The disc is dispensed by the machine

17. (Extension Point: **Send receipt by email (id 5)**)

18. Use case ends

Alternative courses

Step 1. Customer clicks on rent with unlimited Movies button. The use case resumes at main ‘success’ scenario step 2.

Step 4. There are/is no disc(s) for the movie selected. Use case resumes at main ‘success’ scenario step 2.

Step 7. System allows customers to enter a CODE to get a discount on a rental movie. For example, If the customer use the code FREE123 and it is her/his first transaction, the customer will get her/his first disc rental free. Then, the system show details about the rental and use case resumes at main ‘success’ scenario step 8.

Step 11. Card can not be read. Customer removes the card from the reader. Use case resumes at main ‘success’ scenario step 18

Step 13. Card is not enabled to use by the bank. Customer removes the card from the reader. Use case resumes at main ‘success’ scenario step 18

Step 13. Pin entered is incorrect. Customer removes the card from the reader. Use case resumes at main ‘success’ scenario step 18

Step 14. Customer does not exist in the database. System processes the payment, then a new user is created and details about the rental are saved in the database. Use case resumes at main ‘success’ scenario step 15

Step 14. Payment processing fails, use case resumes at main ‘success’ scenario step 18

Step 14. Customer has an unlimited movies account, so the payment is not processed (because the rental is for free) and the rental details are saved in the database. Use case resumes at main ‘success’ scenario step 15

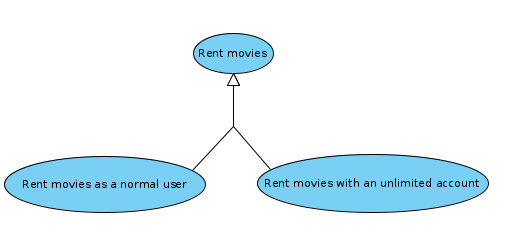
Step 17. **Send receipt by email** fails, use case resumes at main ‘success’ scenario step 18.

Step 16. The machine is broken, the movies was not dispensed, an email is sent to the Xtra-Vision customer service. Customer is also informed about this issue. Contacts are shown to the user. Use case resumes at main ‘success’ scenario step 18.

Priority: High

In this use case, I described as an alternative courses that a user, who has an unlimited account, can rent a movie. The steps vary slightly from the 'success' main path, That is why I documented these minor variations as alternative courses in the use case description. Another option would be, to create an inheritance relationship among use cases to model this specialization.

I could have created three use cases:  
1. Rent movies  
2. Rent movies as a normal customer  
3. Rent movies with an unlimited account.



The rent movies use case would be abstract and would describe the general steps. Then the use cases 2 and 3 would add their own steps to the general behaviour. As these differences are not significant, I chose the option implemented. If I had chosen the inheritance model, I could add more details to every use case, for example, to validate if the quantity chosen is less than or equal to four or two. Four for customers who do not have an unlimited account, and two for customer who have it.

I decided to draw only one actor (I could have created another one called customer with unlimited movies account) because I want to show every customer (new or not) can start the use case, the only difference is at the end when the checkout is done, at that moment the user account is determined..

id: 4

Use Case: **View Movies**

Precondition: The list of movies need to be up to date

Actor: Customer (User)

Goal: Customer wants to see all the movies to rent

Overview: Customer starts to operate the machine, then click on rent button to see all the movies

Cross-reference: R7

Typical course of events:

Actor action System response

1. The customer taps on rent button

2. Displays all the movies with the most important information about the movie (title, genre, how many discs are in the machine, year, etc.)

3. Use case ends

Alternative courses

Step 3. Click on a movie to displays details about the movie such as description, trailer, genre, availability, how many discs are in the machine, etc. Use case resumes at main ‘success’ scenario step 2

Priority: High

---------------------------------------------------------------------------------------------------------------------

id: 5

Use Case: **Send receipt by email**

Precondition: The payment was processed successfully

Actor: Customer (User)

Goal: Customer wants to get the receipt by email

Input: Receipt and email

Overview: After customer pays, (s)he is prompted to type an email address, as a result, the receipt is sent to the customer

Cross-reference: R3

Typical course of events:

Actor action System response

1. Asks the customer to type the email to send the receipt

2. The customer keys her/his email address

3. Asks customer for confirmation, in other words, if the email address entered is correct

4. Customer confirms the email

5. Registers the email in the database for this customer

6 Sends the receipt to that email address

7. Use case ends

Alternative courses

Step 1. The system has an email address for this user/customer in the database. Use case resumes at main ‘success’ scenario step 6

Step 3. Email typed is not valid. Use case resumes at main success scenario step 2.

Step 6. System fail to send the email, retries step 6 or informs user/customer about this issue and use case resumes at main ‘success’ scenario step 7

Priority: High

**---------------------------------------------------------------------------------------------------------------**

id: 6

Use Case: **Checkout**

Precondition: The rental machine works with the type of card inserted by the client

Actor: Customer (User)

Goal: The total cost is paid with a credit/debit card by a customer

Input: Total cost

Overview: Customer pays the total cost using a credit/debit card

Cross-reference: R3, R8, R10

Typical course of events:

Actor action System response

1. Asks the customer to insert the card all the way into the reader

2. The Customer inserts the debit/credit card into the

chip and pin reader

3. Verify card and asks the customer to insert her/his pin number

4. Customer inserts the pin number

5. Verifies card and pin number

6. Processes the payment with the final total cost.

7. Customer removes the card from the reader

8. (Extension Point: **Send receipt by email (id 5)**)

9. Use case ends

Alternative courses

Step 3. Card can not be read. The use case resumes at main ‘success’ scenario step 9. An email is sent to the company showing this error.

Step 5. Card is not enabled to use by the bank. Use case resumes at main ‘success’ scenario step 9. An email is sent to the company showing this error.

Step 5. Pin entered is incorrect. Use case resumes at main ‘success’ scenario step 9. An email is sent to the company showing this error.

Step 6. Payment processing fails, use case resumes at main ‘success’ scenario step 9. An email is sent to the company showing this error.

Step 8. **Send receipt by email** fails, use case resumes at main ‘success’ scenario step 9. Systems lets customer knows about this issue. An email is sent to the company showing this error.

Priority: High

**---------------------------------------------------------------------------------------------------------------**

id: 7

Use Case: **Return movies**

Precondition: The cases are properly closed, and disc/cases belong to Xtra-vision.

Actor: Customer (User)

Goal: Customer wants to return movie(s)

Overview: Customer starts to operate the machine, and (s)he inserts the case into the return slot. Then, if the customer is late returning the disc, (s)he must pay the difference. Finally, customer pays, and then leaves.

Cross-reference: R1, R4, R10

Typical course of events:

Actor action System response

1. The customer taps on Return button

2. Opens return slot

3. customer inserts the disc into the slot

4. The use case resumes at main ‘success’ scenario step 2, if there are more discs to return

5. Gets customer details and Rent details. Then, checks customer returns the disc on time. In other words, There are no late fees

6. Updates rental details

7. Use case ends

Alternative courses

Step 4. Customer is late returning the disc(s), system lets customer knows about it. The system show details about the late rental return. Then Initiate **Checkout (id 6)**. The use case resumes at main ‘success’ scenario step 6.

Priority: High

Use case must be easy to read, so I’ve been working on this during the whole use case work. I chose this template because the division Actor action on the right and System response on the left gives readability to the description.

**4. A Class diagram that will properly illustrate the relationships between classes.**

We drew two diagrams. The first diagram (class diagram 1) shows the attributes and methods of each class, the formal parameters (in) and return types for each method. Additionaly, we drew the relationships between the classes. This diagram displays design/implementation details such as methodPayment class which is implemented using the strategy pattern design (<https://dzone.com/articles/java-the-strategy-pattern>). Also, this model shows details about the design of the software, such as access modifiers (private, public and protected).

Another other important point is, a Rental class can have more than one offer (0..\*), so a new customer could get two special offers, for example customer can get a movie for free using the FREE123 code, and 2 movies for 2 nights. I added \* in the multiplicity, because if we want to extend the system to accept more offers, the system should be flexible to work with more offers.

On the other hand, the second class diagram (class diagram 2 (with more details)) contains more details about the design of the solution. We also added new inheritance relationships, as well as packages. There are four packages, the first one includes all the storage mediums, brands, and the machine. The second one, everything related with the rental. The relationship between MaintinSMList and Rental packages is through Rental class, so there is one point of connection between them. As Detail class is part of Rental (composition relationship), every class must send a message to Detail class through Rental Class. Then, there is another package which just includes the Customer class, and the last one is payment with all the classes related with a payment.

The Rental class contains data about prices and late fees. We added these attributes to this class because we want to keep record of the unit prices in the rental class, and we use the class Payment just to print receipts and process the payment. We also could save information in Payment class about taxes and currencies, if we needed to add this. We could have designed this system using a Details class for Payment class to save everything related with the cost, but the design would have been very big.

**5. A Sequence diagram that will properly illustrate the flow of the system functionality.**

The sequence diagram created does not show so much detail because we are using this diagram to create the functional analysis of the system.

**Rent two movies**

The scenario I chose to create the sequence diagram for the ‘rent a movie’ use case is a customer who selects 2 movies (2 nights) for 2.99 each. The customer pays with a debit card and (s)he is already registered on the system. In other words, the user have already rented before. The email was already loaded in the database, so the receipt is sent to that email automatically. Operations such as prepareCardReader() and releaseCardReader() are not used in this diagram because we do not want to add complexity. These methods are displayed in the next diagrams. CalculateTotalCost() message calculated the total cost of the rental, and it just displays how to get the price from the first movie. The process to get the price for the second movie is similar to the first one.

Lastly, the scenario which a customer has an unlimited account and wants to rent a movie is very similar to this one. The differences would be:

1. We will create an Offer object to reflect the unlimited account (price 0 and unlimited days)

2. CalculateTotalCost() will return 0 because the user has an unlimited account

3. Checking the quantity of discs rented previously plus the discs selected is less than or equal to 2

**Return a movie**

The scenario for this drawing was a customer who return a disc late. (S)he paid with a debit card the overdue, so the customer does not have an unlimited account, and the receipt is not sent because the customer does not want to get it by email.

On the other hand, we added to this diagram the methods to interface with the machine (hardware). These belongs to the machine class. It act as a facade with the hardware. This class could have more than one responsability (verifyCardAndPin() and OpenReturnSlot()), but my objective was group all the functionalities related with the hardware ‘library’ in one class. For instance, we could have added another class called ReaderCard to encapsulate the method verifyCardAndPin().

VerifyCardAndPin() method verifies the PIN which is saved in the card. So it goes as fast as the chip run.

**Open an unlimited movies account**

The scenario is simple: a customer, who does not have an unlimited account, wants to open one to rent. (S)he paid with a debit card the plan, and then the receipt is sent by email to the address registered on the database.

Again, methods from machine class were not added to this drawing. The parameters unlimitedAccount and endUnlimitedAccount must be set at the end, after the payment is done.

**Close unlimited movies account**

Customer close an unlimited account and did not get the receipt by email with amount zero (0), because the user does not want to add it.

**6. A State Diagram modelling the state of the system as a movie gets rented**

We added a state diagram for the Details class because this class behaves dynamically. For example, if the customer is late returning a movie, Rental class will send a message to this class to get the value of late fee attribute. On the other hand, If the customer returns a movie on time, Rental class does not need to know the late fee. So, we believe this dynamic behaviour needs a state diagram to model these situations. We also need to track all the information, when a disc is rented, sold and returned, so having a state attribute in Detail class allows us to saved all this information and query it quickly.

**7. An Activity Diagram which models the entire flow of activity. Include ‘swim-lanes’**

This diagram is hard to understand how it works. What we did was reading about the Petri net, which is a directed graph that uses tokens to represent configurations of the net. In activity diagrams, token contains an object or datum which are input/ouput for/from the actions in the diagram.

There are some parts in the diagrams where we omitted the merge node, because the diagram is clear without this diamond merge symbol.

Some diagrams show datastore to store information persistently. It should be added to all of them, if we want to give more details about operations. We are doing an analysis of the system. The idea of this diagram is to offer useful information for the developer. Information which helps the developer to understand the process. Therefore, this diagram describes a use case with more details.

Verify card and pin action contains an activity inside, so it is a subsection. It needs the input Debit/Credit Card object to execute this activity. It should be modelled inside of the sub-activity, but because of simplicity, we drew the arrow from the object to the sub-activity.

**8. One User Story, following standard Agile practices, that details a user interaction feature of the Xtra-Vision system that you would like to have included in a future version that is not currently part of the original system description**

As a customer, I want to know where I could rent a movie if there is not stock in the machine for that movie so that I can go to that location and rent that movie .

- The tables of the database, which saves information about the machine and discs inside of that machine, must be updated when a customer rents/returns a movie.

**Research item 2: Dealing with design complexity: Software systems can start off simple and quickly grow to be quite complex and therefore when developing a computer system, the developer will often be dealing with a certain amount of complexity. Explain (in your own words) how modelling techniques manage complexity in the development of computer-based systems. Your answer should include a description of how models can support software developers who are striving to understand the complexity of the systems that they are developing. Be sure to include factors that are typically used to deal with complexity when analysing a system during the UML modelling process (research and references required).**

We have two different kind of software, those which can be developed without a previous analysis and design, and those which need a deep analysis and design because they are considered complex. When we say complex, we refer to the complexity of the domain problem, the difficulty of managing the development process, the difficulty of developing an extensible software and the difficulty of analysing/designing computer systems.

Nowadays, there are complex software which have millions of lines of code, and these one require UML models to deal with their inherent complexity. These models reduce the complexity because we can analyse the software from different views (user’s and developer’s view), we can use them to interact with end users and getting feedback quickly, we can divide the problem into two or more simpler sub-problems to have a better understanding of the original problem, we can use a model to feed activities in the development life cycle, we can create the structure of the system, show its behaviour and describe the process fully, etc. As we can see, UML models give us many advantages for building software.

Models are abstraction of computer-based systems with tools such as abstraction, encapsulation, descomposition and hierarchy to deal with complexity of the software, and thus creating simple representations of the system.

We are going to explain every concept and give some examples from the object-oriented paradigm.

**Abstraction** means hiding the details. We are using abstraction when we want to identify the required characteristics of an object/class ignoring the details which are not important. For example, if we are working on a system to rent cars, we can focus on the different models and characteristics of cars and ignore details about how the motor was built.

We focus on grouping, generalizing and chunking characteristics of different entities to built an abstract model. Another example:

We want to model a system to rent cars. We know the company has several types of cars which have common characteristics, so we can identify these characteristics and group them in an abstract class. Then, the concrete classes inherit the attributes/operations from the abstract class using the ‘is-a’ relationship.

**Descomposition** is based on the idea of divide and conquer. We can break down a complex system into smaller simpler parts or subsystems, to understand them separately. Then, we do the same with the new subsystems, until we can get a solution for every part. As a result, we will built a solution for the whole system.

There are two types of descomposition, process-oriented and object-oriented.

The first one is about dividing the whole process into modules/programs. Modules represent functions, which can be divided again. The second one, which is more popular nowdays, divides the systems into classes which encapsulate a state and a behaviour.

**Encapsulation** is data hiding. We can encapsulate data and functions inside of a class. So, from outside the clients do not know how it is implemented, they just can call the functions from the interface. It favours the extensibility of a system.

**Hierarchy** allow us to make relationships among objects/classes. With relationships we can group/separate the responsabilities and properties of the objects, and thus makes the system easier to understand. For example, the ‘is-a’ relationship classify the objects into groups, and thus we can distinguish the common attributes/operations of the objects. Another relationhip is ‘part-of’ which allow us to define objects as part of another object, for example a car has wheels, motors, doors, etc.

All the tools, which can be modelled in UML, make easier the building of software. UML models help to build diagrams which are platform-independent describing the functionalities and the system behaviour. More details we add to the diagrams, the work of the developers will be easier, because they will get models which will provide implementation decisions. For example, activity diagrams allow us to describe the details of an operation, class diagrams can include interfaces, abstract classes, the visibility of attributes and operations, controller classes, operations in detail, GUI classes, etc.

As a conclusion, we can say a developer can not hold all the details in his head about a complex system, they need to break down the problems into smaller ones and analyze them separately. They also need to create UML diagrams to show the structure and behaviour of the system using classes which are abstractions of objects in the real world. By applying the concepts described previously, we can define an abstract model of the software, which is a simpler representation of the software. This representation will be used later for the implementation/coding of the system.

**Report and Enhancements**

The project was divided in two parts, the first one is related to the gathering of the requirements and analysis of the system, and the second one to the development of the prototype.

How did we work in the first stages of the development?

We used diagrams and visual paradigm tool to analyse the system. We read the book “A Student Guide to Object-Oriented Development” and we did a research about the Visual Paradigm tool to understand how it works.

Prior to doing the diagrams, we created the requirements, defined the scope of the project and planned the time it would take to get the project done. We knew this project was going to be big enough to not finish it, so we focused on the analysis to create a design which helps to develop the prototype quickly. Then, everyone designed the same diagrams, and after that we met to discuss the design decisions taken in the diagrams. Finally, we created the final diagrams together with visual paradigm.

How did we work in the last stage of the development? (creation of the prototype)

The prototype was created using Neatbeans IDE 8.2 and the object-oriented programming language Java version 1.8. We are going to describe the steps performed to develop the system:

1. We created the database in MySQL Workbench using the class diagram. We translated the class diagram in an Enhanced Entity-Relationship (EER) diagram using the features provided by the visual database design tool. We thought we could not draw the entire database, so we created some objects in memory to save time. For example, movies, some attributes of discs and machines are created when the system is executed. Attributes which can be changed in runtime and affects the next runs are saved in the database. Rental, details, customer and some attributes of discs are in MySQL. We did this step together.

2. We created the view class, which is used by the controller class. The only knowledge that controller class has about view class, is what view class has exposed through its interface. Therefore view and controller are loosely coupled. We used the use case descriptions to “draw” the view, because these descriptions were described from the users point of view. Then, in the controller class we defined the menu and created the messages which would be displayed to the user. Lastly, We tested the view class. These tests were more technical than functional (about the internal process). We did this step together.

3. We had the class diagram divided by packages, so what we did was splitting the coding task. One team partner developed two packages, and the another one the two others. Then we met to see the code together and to work together.

We dumped the class diagram into Java, so we defined the classes, attributes and relationships. Some of the relationships were not necessary, so we did not add them. Sequence diagrams and activities diagrams were a roadmap to guide us through the sequence in which the messages from the objects should be executed. The state diagram was used to code the behaviour of the classes Detail and Disc.

4. We added the library mysql-connector-java-8.0.24 to the project to connect with the database, and then we implemented all the methods from the diagram class. For example, getAll(), get(), save(), update(), getQuantityOfRentalsNotReturnedByCustomer(), etc. We worked together in this class.

5. We worked together to connect all the parts/packages/classes in the controller class. We used the use case diagram and the gathering of requirements to check all the functionalities were implemented. The return functionality generates a returning automatically using the class Random. In other words, the system chose a disc randomly to return.

6. We added the libraryjakarta.mail-1.6.6 to code the issueReceipt() method. We did a research on internet to find ways of sending an email. The method implemented seemed the quicker algorithm to send an email.

7. We tested the whole process together and we document the testing in an excel file. First, we tested technical issues, and then the internal process using the use case descriptions to create tests.

Lastly, we updated some details in the diagrams in order to make the analysis and the prototype consistent. The analysis is also used as a guide for analysts, designers, developers, etc; if they want to understand how the system works.

Enhacements:

1. There are some fields in the database which can be null, so another solution could have been: set the fields lateFee and dateReturned, when the rental is done, with data which would be inserted if the customer is not late returning the disc. We saved operations and processing time.

2. We could have created an inheritance structure for the customer class. We may have two classes, CustomerAccount and CustomerUnlimitedAccount. The second one would inherit the attributes from the first one, and we would add the dates (startAccount and endAccount) in CustomerUnlimitedAccount class. With this structure, the implemented validations would have been simpler to understand.

3. We did not save the time when the customer returns a movie. It should be added as well.

The file 05112021.sql is the most stable version of the database. In case, you have a problem you could load this database to the server.

References:

1. Britton, C. and Doake, J., 2005. *A Student Guide to Object-Oriented Development*.

2. Cossentino, M. *Using UML, Patterns, and JavaObject-Oriented Software Engineering*. [ebook] Available at: <http://www.pa.icar.cnr.it/cossentino/se15-16/ppt/L2\_ModelingwithUML\_ch02lect1.pdf> [Accessed 19 April 2021].

3. n.d. *Object Oriented Analysis & Designusing UML*. [ebook] Department of ComputerScience and Engineering. Available at: <https://vsm.edu.in/R-16-CSE-III-I/OOAD.pdf> [Accessed 19 April 2021].