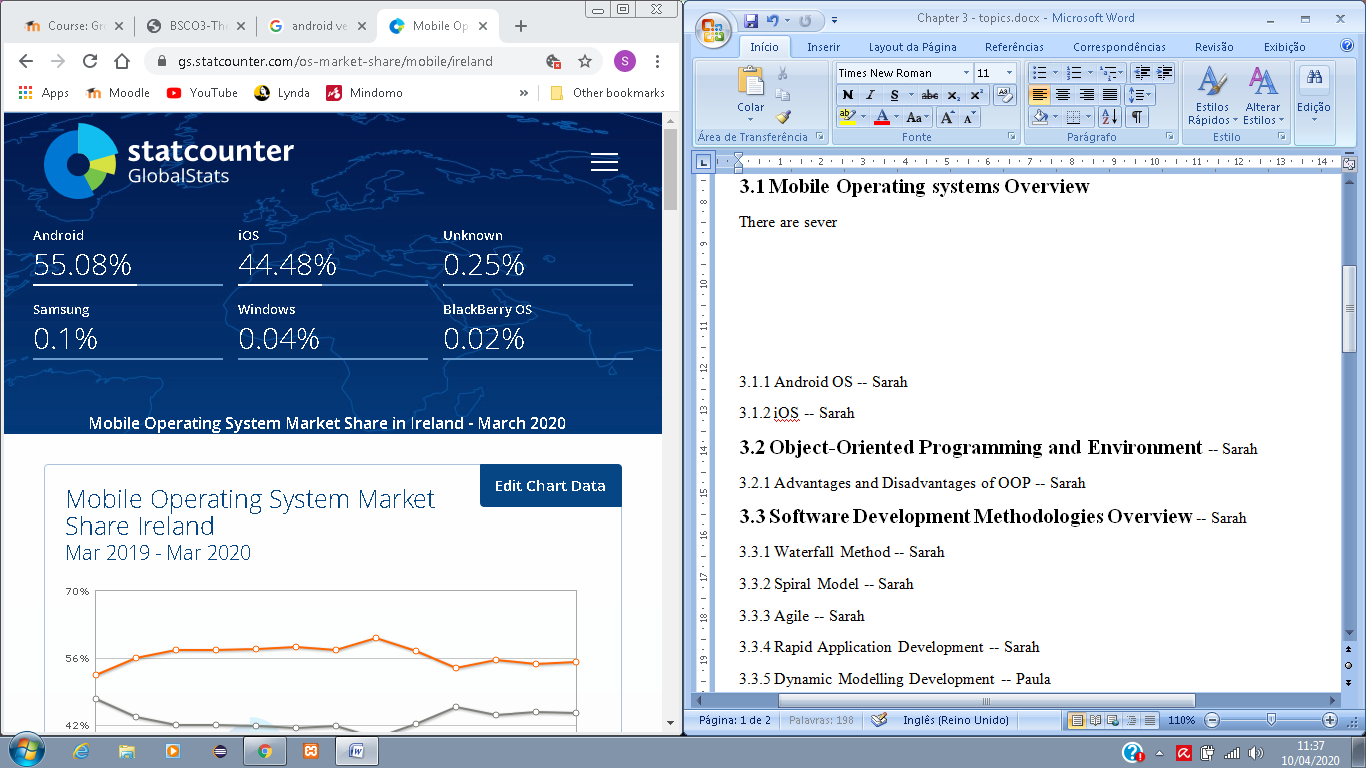
# Chapter 3: Project Application Analysis Phase

The Secondary and Primary research carried out on the topic and discussed in the previous two chapters, have proved the feasibility of and the need for development of a suitable mobile application to help potential adopters in Limerick, Ireland, find a desirable pet to love. In order to do so, an appropriate Development methodology must be implemented. This chapter describes some of the Development methodologies used in the industry today and compares their advantages and disadvantages in the process of selection of most appropriate for the current Project methodology to be followed during the Project’s development stage.

## 3.1 Mobile Operating Systems Overview

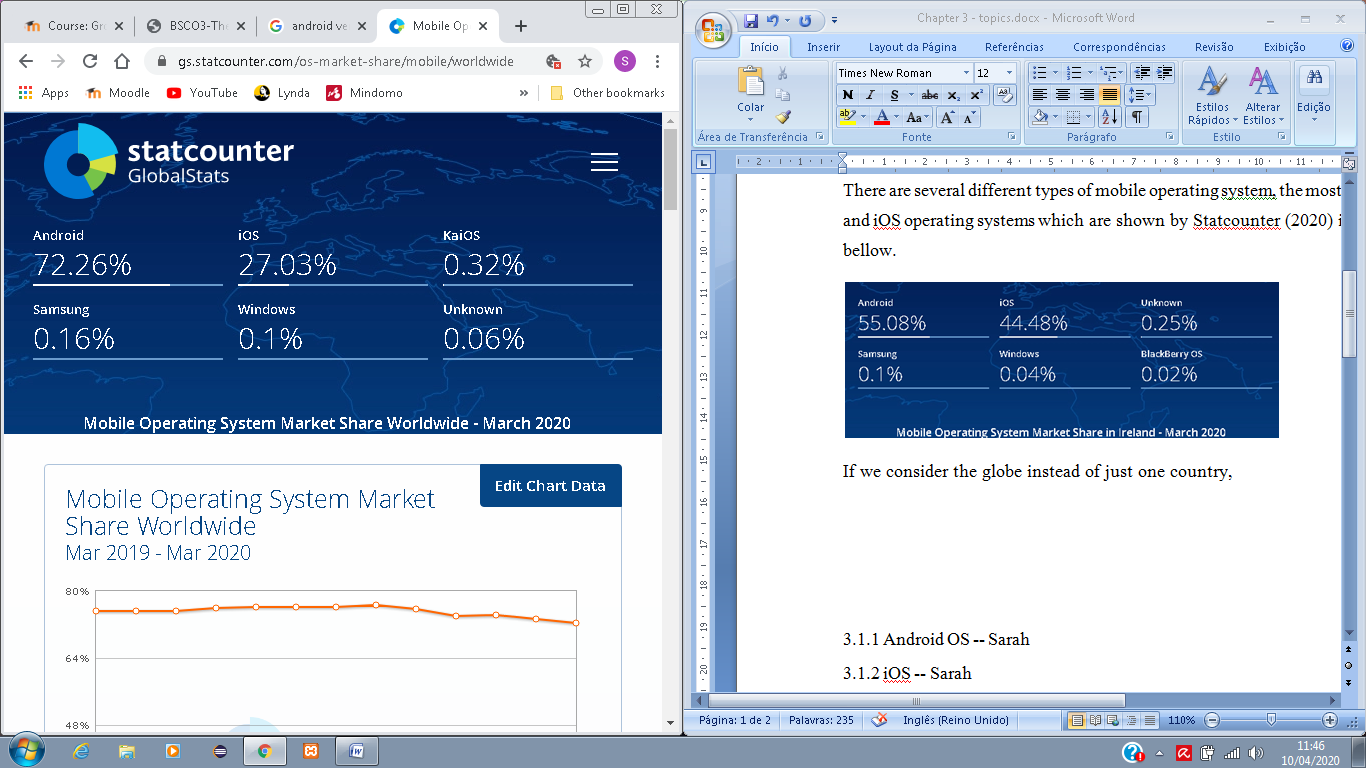
There are several different types of mobile operating systems (OS), however, the most used OS in Ireland are Google’s Android and Apple’s iOS, as shown by StatCounter (2020a) on Figure 3.1.



*Figure 3.1: Mobile Operating System Market Share in Ireland - March 2020*

*(StatCounter, 2020a).*

If we consider the worldwide view, instead of just one country, the research still shows a greater frequency of Android and iOS OS and a dominance of Android over iOS (please, see Figure 3.2).



*Figure 3.2: Mobile Operating System Market Share Worldwide - March 2020*

*(StatCounter, 2020b).*

Having said that, it is more than reasonable that Android and iOS were the options to be considered for the current Project application, since the other alternatives do not show considerable market share, thus it would not be worth the application development time for them.

Ideally, such application as for the current project, should be developed for both OSs that are most widely used, in order to reach a wider target audience, however, due to time restrictions only one will be implemented in this current project, with the Android OS being the chosen one, since is the most worldwide used (with 72.26% of users globally), and also for reasons like better programming language alternatives, IDE tools, Emulators and the verification procedures that are more accessible by *Google Play* Store than the *Apple App* Store. These characteristics of Android Operating System will be discussed in the next section.

### 3.1.1 Android OS

Rich Miner, Nick Sears, Chris White and Andy Rubin founded the company Android Inc in 2003 in Palo Alto, California. By that time Rubin said that they would develop smarter devices that would be more aware of its owner’s location and preferences; this was the first idea for the *Android* OS. After that, the company and the software itself have undergone a lot of changes, including the acquisition of *Android Inc* by *Google* in 2005. *Android* is an open source Operating System with a great user base and a simplified mobile app development process (Sharma, 2019). All the *Android* versions, since 2008 when its first public debut was made, are listed as follows (Callaham, 2019):

* 2008 – Android version 1.0 to 1.1
* 2009 – Android version 1.5: Cupcake (birth of Android version names tradition)
* 2009 – Android version 1.6: Donut
* 2009 – Android version 2.0 to 2.1: Eclair
* 2010 – Android version 2.2: Froyo
* 2010 – Android version 2.3: Gingerbread
* 2011 – Android version 3.0 to 3.2: Honeycomb
* 2011 – Android version 4.0: Ice Cream Sandwich
* 2012 and 2013 – Android version 4.1 to 4.3: JellyBean
* 2013 – Android version 4.4: Kitkat
* 2014 – Android version 5.0 to 5.1: Lollipop
* 2015 – Android version 6.0: Marshmallow
* 2016 – Android version 7.0 to 7.1: Nougat
* 2017 – Android version 8.0 to 8.1: Oreo
* 2018 – Android version 9: Pie
* 2019 – Android version 10

Version 10 is the most recent version of *Android* and is the version that quit the dessert themed tradition, by being known simply by a number (Raphael, 2020).

There is a range of Integrated Development Environment (IDE) and programming languages that can be used when developing for *Android* OS. Starting with the programming languages, *Java* is the most popular one. However, *Kotlin* is an alternative to *Java* since it is easier to read and code for developers, especially for beginners, which means that coding can be done in a more efficient way. C# is another alternative, which is supported by tools like *Unity* and *Xamarin* for gaming and multi-platform applications (The Tool, 2018). According to The Tool (2018), there are different tools that act as Front-end Application Development Environments (IDEs) for Android, including *Eclipse*, *Android Studio* and *IntelliJ* IDE’s.

*Kotlin* and *Java* are both open sources and Object-Oriented programming languages. The first one, although easier, “better language” than *Java*, that is sponsored by *Google* and been announced one of the official languages for Android Development in 2017 (GeeksforGeeks, 2020), is not going to be used in the current project, due to limited time for project development and lack of experience of the development team with it. Since *Java* is the language we are more familiar with, it has been the chosen programming language to use in the current project.

## 3.2 Smart Home Devices Overview

In a world where everything at home is smart, or is capable of being smart, it is important to mention the devices that allow the interaction with other devices in order to maintain the quality of smartness. *Smart Home* devices are IoT devices that connect with other devices, such as bulbs, cameras, sensors, etc, and allow full control of those devices by the users, according to their needs. Even though *Apple* is in the market and has made improvements over the past few years with Apple *HomeKit*, *Amazon* and *Google* hold the supremacy of the market with *Alexa* and *Google Home* (Reisinger, 2020).

Both devices, *Amazon Alexa* and *Google Home*, present the same abilities, such as allowing set up of alarms, control of other smart devices, listening to entertainment, etc. However, certain attributes are more specific to each one of them. *Alexa*, for example, has a wider range of Third-party support for connected devices, but there are some restrictions on skills if the user is not an Amazon Prime member. *Google Home* on the other hand, comes with its own built-in virtual personal assistant, *Google Assistant*.

For reasons, mentioned in the previous sections for choosing Android Operating System, the developers of the current project’s mobile application have chosen to interface it with the *Google Home*, once it has been developed.

## 3.3 Object-Oriented Programming Environment Characteristics

Object-Oriented Programming (OOP), unlike the procedural programming, uses the principle of dividing the “problem” into small components, which are the objects. Usually, an Object-Oriented program is comprised of many different objects, which can store information, perform certain behaviours, and interact with other elements of the environment (Anderson, et al., 2015). According to Anderson, et al. (2015), these objects are related to real world entities. For example, a person is considered an object, which has attributes such as hair colour, eyes colour, name, etc.; these would be the properties of the object; and also, the person can undertake actions, such as eat, travel, sleep, which would be the methods within the object.

* ***Encapsulation***: is one of the most important concepts in OOP. This concept involves *hiding information*, whereby the object only provides essential information for users to manipulate. The internal details are hidden for the outside world, in order to avoid the data manipulation, and especially to avoid possibility of the user performing something they should not do within the program. This can be done by defining the attributes and the actual piece of code of the methods as private, allowing only the methods name and signature being known by the outside world (Nakov, et al., 2013).
* ***Inheritance:*** with this principle, a class is able to inherit the behaviour and the characteristics of a class that is more general, for example a class for cats has attributes and behaviours, such as having four paws, being predators and hunt their prey. All these characteristics and methods can be coded and be reused in a different class for “Felidae” which is a biological family within the cats that encloses Lions, for example (Nakov, et al., 2013).
* ***Abstraction:*** this is a principle that uses the idea of not knowing how things work internally but knowing how to use them. For example, a remote control for the TV, we do not really know how the control operates internally, but we know how to use it by just pressing the buttons (Nakov, et al., 2013).
* ***Polymorphism:*** with this principle, a method or an ability of object class can be performed in different ways. For example, if we have a class of Big Cats and a method for hunting their preys, which is done in different ways by each cat, the Lion class, derived from the Big Cats class, can override its method by setting that Lion sneaks on its prey, whereas the Cheetah, which is another class derived from the Big Cats class, overrides its method by setting it to outrun its prey (Nakov, et al., 2013).
* ***Association:*** this principle establishes the relationship between two classes that are unrelated to each other. That means that they are *associated* through their objects and each one can exist without the other (Monus, 2020).
* ***Aggregation:*** it involves the HAS-A relationship between the two associated classes. For example, a person HAS-A job or HAS-A car but a car or a job does not necessarily HAS-A person. The relationship is one-directional, which means that only one class is dependent on the other (Monus, 2020).
* ***Composition:*** in composition, two different classes are mutually dependent on each other and each of them cannot exist without the other. This principle involves the PART-OF relationship, for example, an engine is PART-OF a car and vice versa. Thus, if engine stops existing, so does the car (Monus, 2020).

### 3.3.1 Advantages and Disadvantages of OOP

**Advantages:**

* Better developer productivity;
* Lower programming costs;
* Organization;
* Decision building;
* Code reusability;
* Security through data hiding;
* Object co-existence;
* Software with higher quality;
* Easy upgrading from small to large systems.

**Disadvantages:**

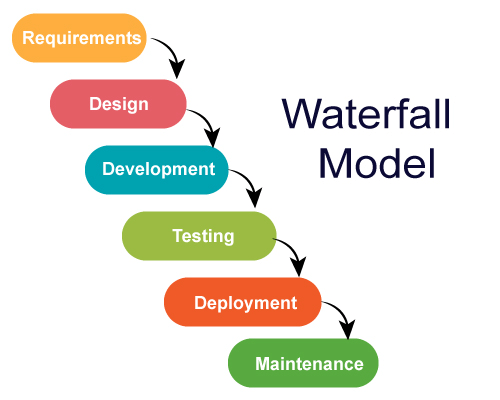
* Slower software;
* Adaptability to Object-Oriented, abstract thinking for programmers;
* Mastery in software engineering and programming methodology is required;
* Difficulty tracing and debugging message passing between objects.

## 3.4 Software Development Methodologies Overview

Managing projects well is what leads to their success. The choice of a suitable Software Development methodology to work best for a project, allows the project to be managed more efficiently. There are different methodologies, with different strengths and weaknesses each, but all of them lead towards the success of a project. The following sections describe some of the used in industry Software Development methodologies with their characteristics.

### 3.4.1 Waterfall Method

*Waterfall* model is commonly considered the most traditional Software Development method. This methodology was first created by Winston W. Royce in 1970, and is a strict linear approach, in which the requirements from stakeholders and customer are gathered at the beginning of the project. A sequential project plan is then created, in order to accommodate these requirements (Project Manager, 2020). There are usually five to seven stages in this model, and a new stage of the project can only start when the previous stage has been fully completed and documented; however, there is no possibility of going back to a previous stage (IntellectSoft, 2019). The stages of this model are presented on Figure 3.3.



*Figure 3.3: Visual Representation of the Waterfall Model (JavaTPoint, 2018).*

**Advantages:**

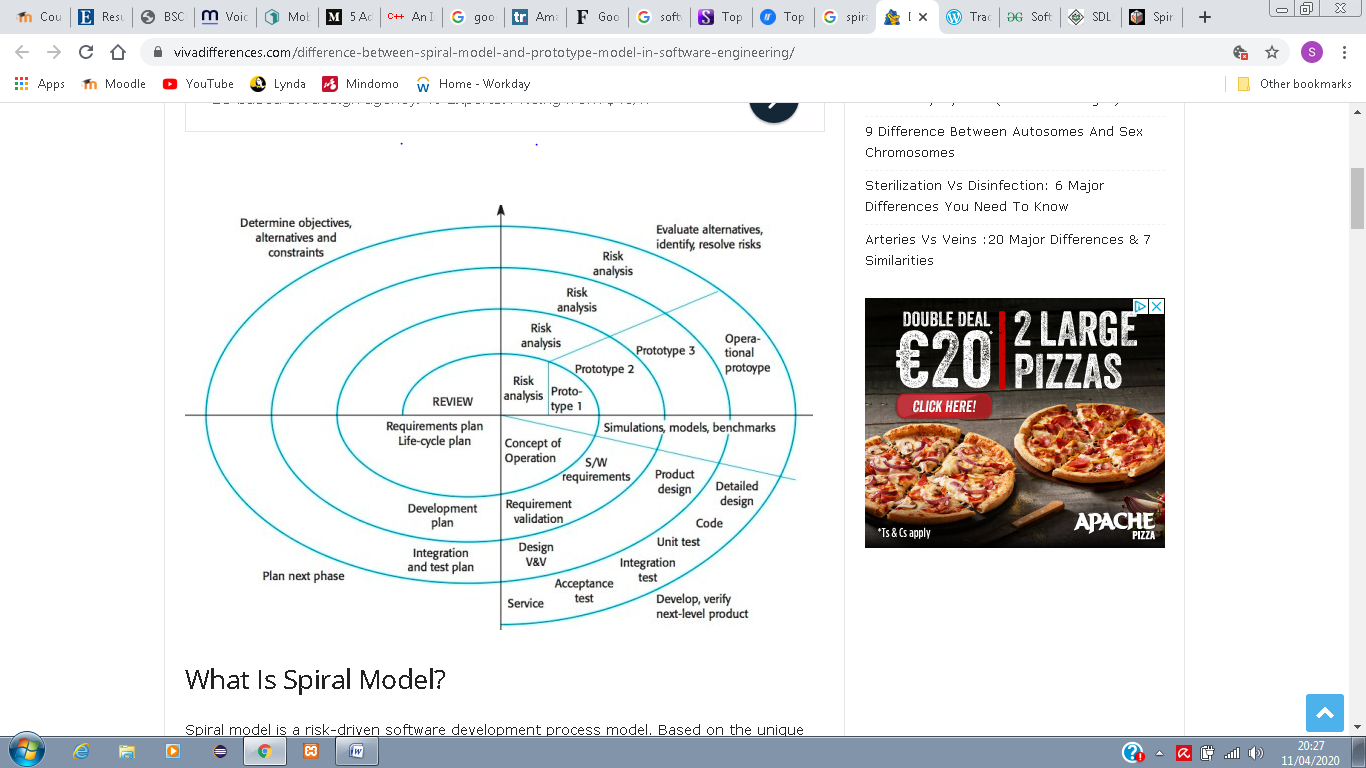
* Simple and clear planning and scheduling;
* More accurate estimates of cost, resources and time schedule for the project;
* Clearly defined stages;
* Well understood milestones;
* Great documentation of process and results;
* There is no delaying of production, since there is no adding of new requirements from customers until product is developed.

**Disadvantages:**

* Customers may find difficulty articulating their requirements at the beginning of the project;
* Can be very expensive to go back to designing the code again if the customer is dissatisfied with the developed product;
* Working software is only produced in late stages within the Life Cycle;
* High risk and uncertainty;
* Not recommended for complex projects in Object-Oriented environment;
* Not suitable for projects with frequent changes in requirements and ongoing maintenance;
* Lacks flexibility in case of unexpected events (IntellectSoft, 2019).

### 3.4.2 Spiral Model

The *Spiral* methodology is a combination of *Sequential* and *Prototyping* methods and is presented like a spiral with several loops. These loops can vary in number, depending on each project, and each of these loops represents one Prototype version of the Project. The Project Manager decides how many Prototype versions are needed to develop the Project, depending on the Project risks. The cost of the project is represented by the radius of the spirals and the progress made so far in the current Prototype version is represented by the angular dimension (Pal, 2019). The phases of this methodology can be seen on Figure 3.4.



*Figure 3.4: Visual Representation of the Spiral Model (Viva Differences, 2019).*

**Advantages:**

* Fast development;
* Strategic ways are created for handling larger projects;
* Proper Risk Evaluation;
* Total control in each phase of the Life Cycle;
* Features can be added in a systematic way at each Prototype version;
* Early production of the software;
* Faster implementation of changes with the customer’s feedback.

**Disadvantages:**

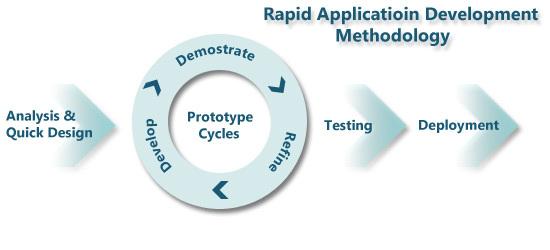
* Expert people are required for Risk Analysis;
* Not recommended for small projects and small organisations, since it may be expensive;
* The Spiral can go infinitely;
* Large amount of documentation (Pal, 2019).

### 3.4.3 Rapid Application Development

Rapid Application Development (RAD) is a pre-Agile project management strategy with a fast project turnaround. RAD is focused on minimizing the planning phase and maximizing the development of the prototype phase. The process can be broken down in a few ways which include four main phases:

* Requirements planning;
* User design;
* Rapid construction;
* Cutover (LucidChart Content Team, 2019).

RAD is a methodology that should be used when the project needs to be developed in a short time frame (2 to 3 months) and can be modularized (Tryqa, 2017). Highly experienced development team is required, since RAD provides a very fast paced environment. The phases of RAD methodology are presented on Figure 3.5.



*Figure 3.5: Visual Representation of the Rapid Application Development Methodology (Islam, 2017).*

**Advantages:**

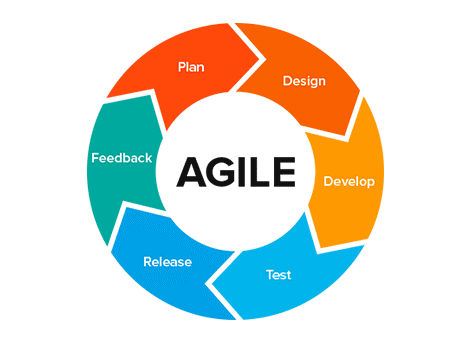
* Risk reduction;
* Increased quality;
* Faster time to market;
* Frequently changing requirements can be accommodated;
* Time for development is reduced;
* Components are highly reusable;
* Initial reviews occur in a quick manner;
* Customer feedback is encouraged.

**Disadvantages:**

* Dependence on strong team and individual developers’ skilful performance;
* Only appropriated to systems that can be modularized;
* Not too appropriated for cheaper projects since the cost for modelling and code generation is high (Tryqa, 2017).

### 3.4.4 Agile

*Agile* methodology was created with the purpose to improve the inadequacies and inflexibility of the traditional methods, like the *Waterfall*. Starting with the fact that software is continuously upgrading, which means that developers also need to improve constantly their skills and be innovative (Muslihat, 2018), the *Agile* software development involves software development practices that are based on iterative development. In this case, requirements and solutions are always evolving throughout the project, with collaboration between self-organizing cross-functional teams. It promotes disciplined process that encourages regular inspections and adaptations, teamwork, self-organization and accountability, and rapid delivery (Sacolick, 2020). It is a framework, divided into number of sub-frameworks, such as Scrum, Kanban, Extreme Programming, Lean Software Development (LSD), Rational Unified Process, Dynamic Systems Development Method (DSDM), among others (Appelo, 2008). The Agile Development process is presented on Figure 3.6.



*Figure 3.6: Visual Representation of the Agile Development Process (Abellán, 2020).*

**Advantages:**

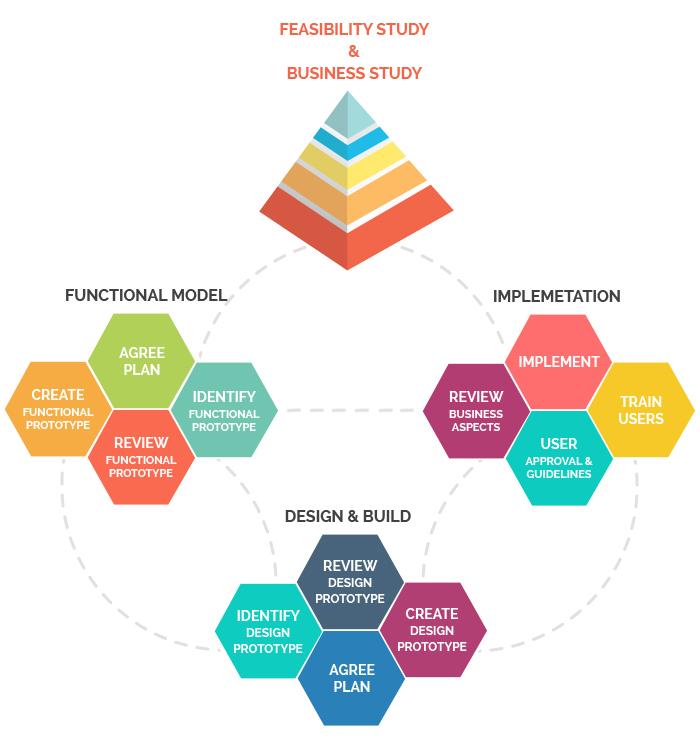
* Flexibility and adaptability;
* Creativity and Innovation;
* Faster time-to-market;
* Lower cost;
* Improved quality;
* Faster bug detection and fixing;
* Customer satisfaction;
* Employee satisfaction;
* Organizational synergy.

**Disadvantages:**

* Required training and high skills;
* Organizational transformation;
* Scalability;
* Integration with project/program management (Appelo, 2008).

### 3.4.5 Dynamic Software Development Method

The *Dynamic Software Development Method* (DSDM), is an *Agile* approach for software development, in which users are required to be actively involved and the decision-making process is made by the team (please, see Figure 3.7). This methodology uses techniques such as *Time Boxing*, MoSCoW rules, and Prototyping (Guru99, 2020). The two main focuses of DSDM model is the strict time frame and assigned budget. It involves continuous feedback in order to deliver maximum functionality, according to the agreed requirements (IntellectSoft, 2019).



*Figure 3.7: Visual Representation of the Dynamic Software Development Method*

*(TatvaSoft, 2015).*

**Advantages:**

* Excellent communication between all involved in the project (developers and users);
* Fast to reach the functionality needed;
* Developers’ easy access to end-users;
* Projects are delivered on time and within specified budget.

**Disadvantages:**

* High cost to implement;
* Developers and users must be trained;
* Not suitable for small organizations (IntellectSoft, 2019).

After considering the discussed above methodologies for software development, DSDM was chosen to be used in this project due to reasons such as time constraints and the skills level of the programmers. DSDM approach uses techniques that are essential for the timely completion of this project, such as Time Boxing, MoSCoW rules and Prototyping. Time boxing is very important in this case because the time management must be very precise in order to deliver the software on time within a very tight timeframe. MoSCoW rules determine what Must Have, Should Have, Could Have and Would like to have, but not necessarily this time, and the Prototyping technique allows the developers to test the software as it happens.

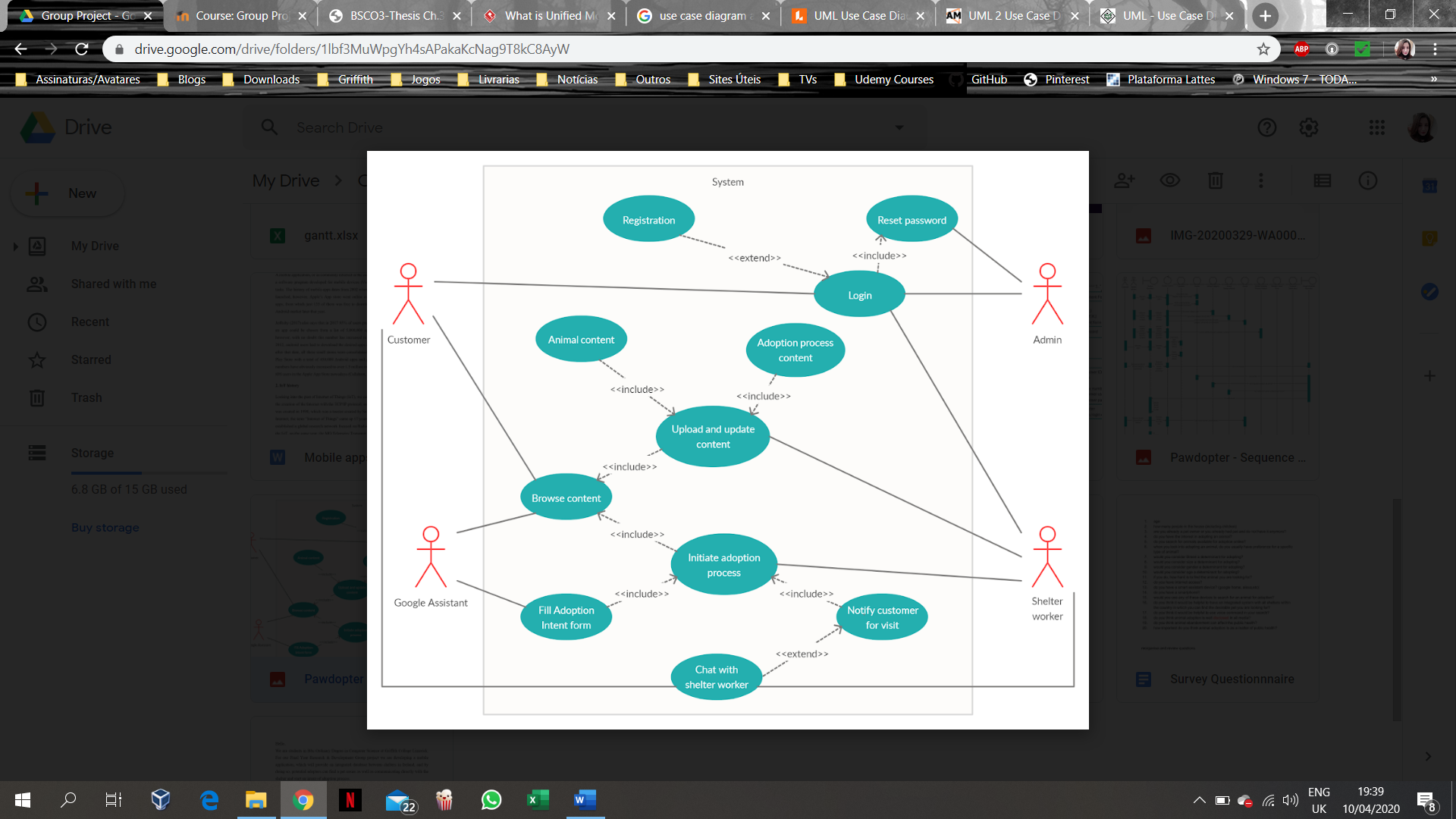
## 3.5 Unified Modelling Language

The Unified Modelling Language (UML) is a standardised language used to specify, visualise, construct, and document the objects of software systems. It was developed in order to help system and software developers by allowing project teams to communicate regarding potential designs of the software (Visual Paradigm, 2020). There are many types of UML diagrams available, but for this project the three that are used are: Use Case Diagram, Sequence Diagram and Object Class Diagram. Use Case Diagrams, as well as Sequence Diagrams, are used to capture the dynamic aspects of a system (Tutorials Point, 2020).

**3.5.1 Use Case Diagram**

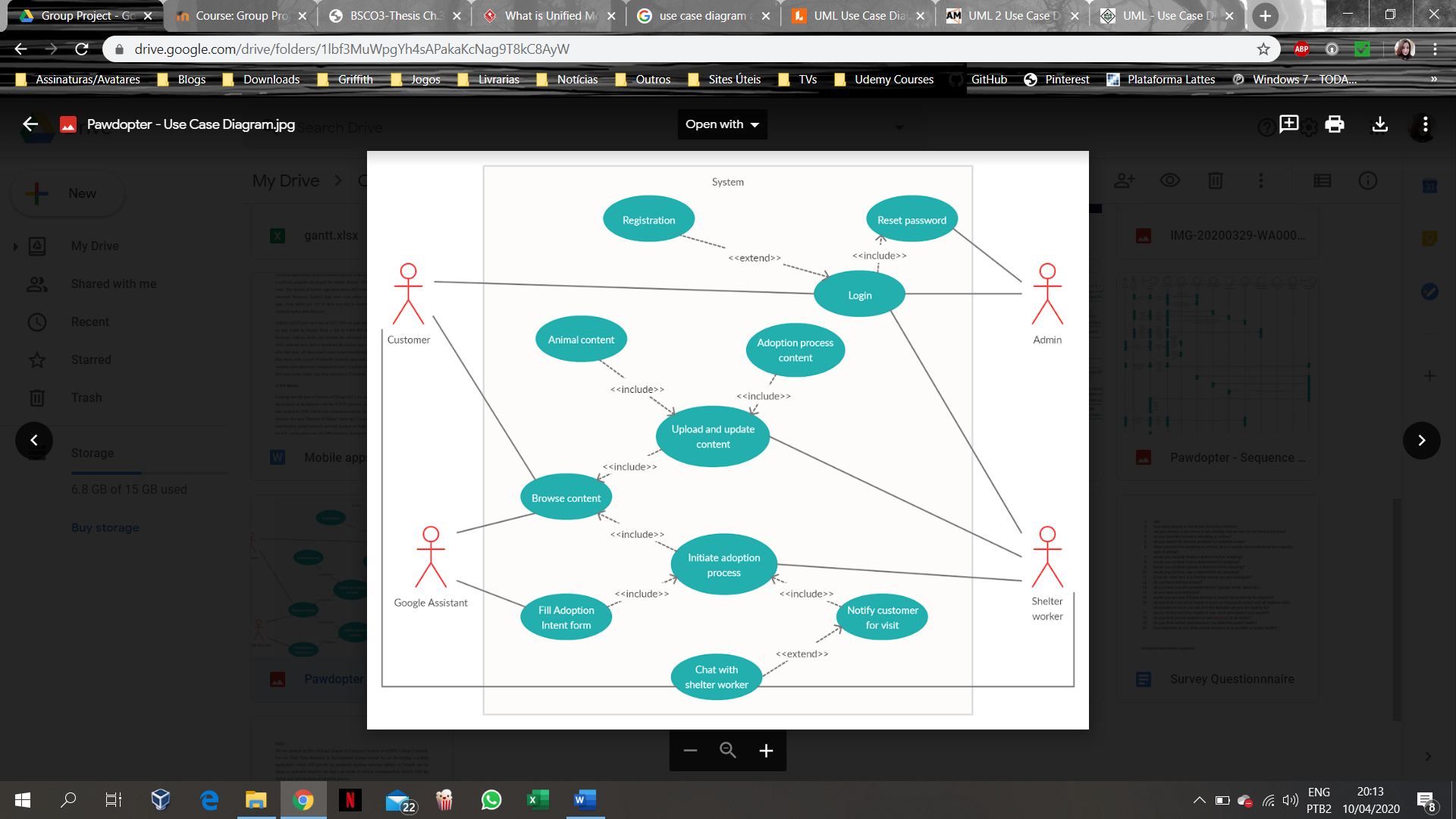
Use Case Diagrams are applied for high-level requirement systems to present all its functional requirements, as well as both internal and external influences, in order to detect all functionalities and identify the actors of the system. They are used to get an outside view of the system and show the interactions between actors and the system and their involvement in the system’s functionalities (Tutorials Point, 2020).

*Actors* in a Use Case diagram can be either human users, or other system(s), external to the system modelled, which are involved directly in one or more functionalities within the studied system. To be an actor within the system the user must produce or consume data (Lucid Chart, 2020a).



*Figure 3.8: Visual Representation of an Actor on a Use Case Diagram.*

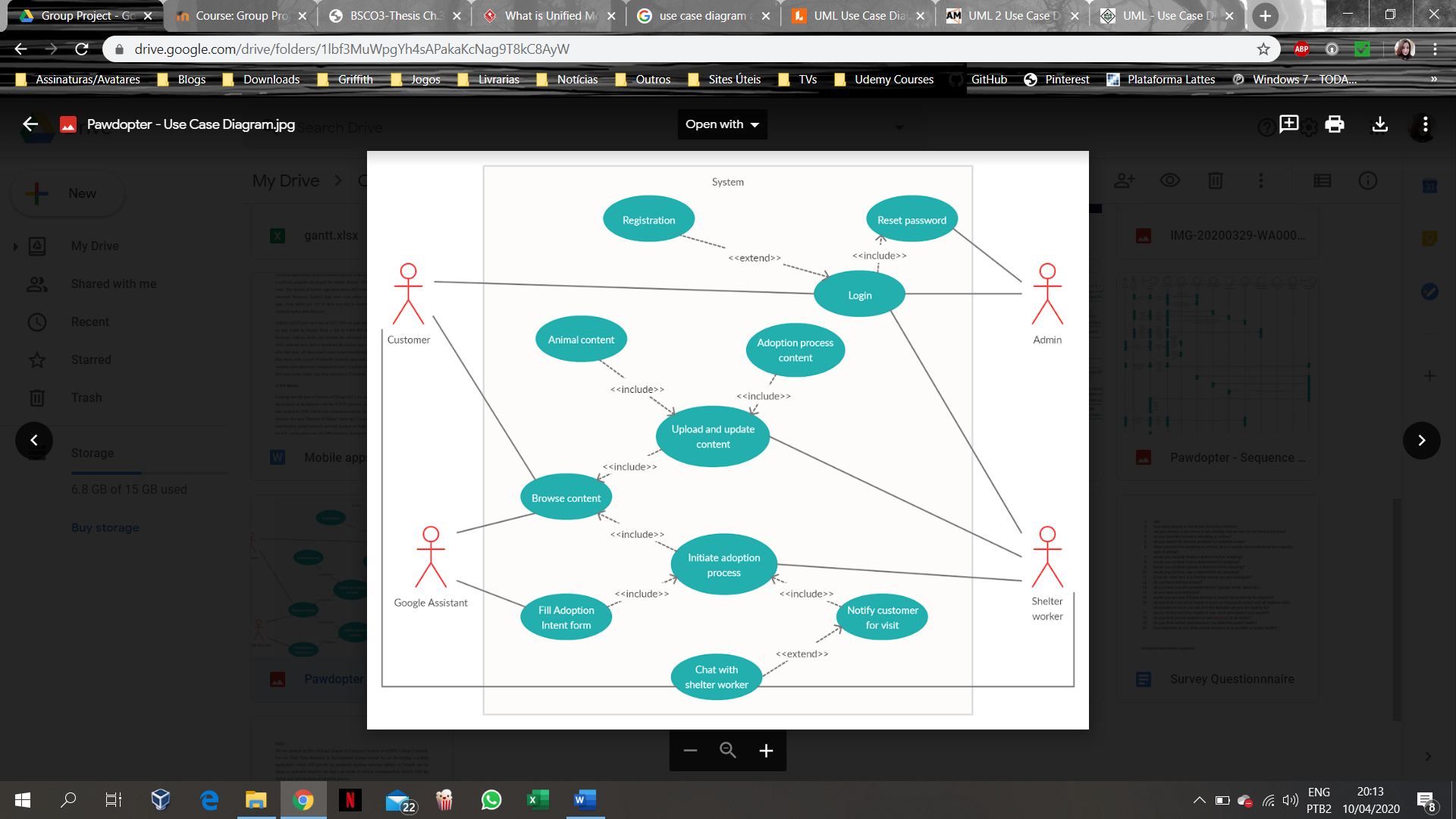
*Use Cases* in a Use Case Diagram represent the functional requirements of the system. The name of a Use Case should be selected in a way that it identifies the functionality to be performed (Tutorials Point, 2020). On Use Case Diagrams, a Use Case is represented by a horizontal ellipse that is going to be connected to an actor and/or another Use Case through a relationship (Agile Modeling, 2020).



*Figure 3.9: Visual Representation of a Use Case in an UCD.*

In a Use Case diagram, a relationship can be presented as existing either between a Use Case and an Actor, or between Use Cases. The relationship between actors and Use Cases is called *association*. It is presented by a straight solid line, connecting an Actor to a Use Case; each actor can be associated with many Use Cases (but it must be associated with at least one), and each Use Case can be associated with many different actors.

The relationship between two Use Cases can be either an <<extend>> or an <<include>> relationship. The <<extend>> relationship takes place when a functionality of the system is extended, adding to the existent one. It is an optional relationship, and the functionality of the *extended* Use Case will be occasionally triggered, depending on a condition. The <<include>> relationship takes place when a common functionality must be reused, usually between multiple Use Cases. It is a relationship that shows that the behaviour of the *included* Use Case (the one, towards the arrow’s head points to) is considered as part of the *including* one (the one, from where the arrow originates). The *included* Use Case presents a mandatory behaviour and can exist on its own, whereas the functionality of the *including* Use Case is incomplete without the functionality of the *included* one (Silva, 2019).



*Figure 3.10: Visual Representation of Relationships between Use Cases and between Use Case and an Actor.*

The Use Case Diagram for the current Project is presented in *Appendix A5* as the High-End alternative solution, designed for the project. It presents 4 different actors outside of the system, which are the Customer, the Shelter Worker, the Admin, and the Google Assistant. There are few use cases that are common to more than one actor, which are *Login*, *Browse* and *Chat with Shelter worker*. *Registration* and *Chat with Shelter worker* are Use Cases that extend functionality of another Use Case (the *Registration* extends the *Login* and it will only be triggered when a new user registers to join the system, while the *Chat with Shelter worker* extends the *Notify Customer for Visit)*. The analysis of the possible alternatives will be discussed further in this chapter.

The *Login* functionality will allow both, the Customer and the Shelter Worker, to enter the system, and be involved in different functionalities, depending on the type of user they are:

If it is a Customer, they may create/update personal *Profile*, and *Browse Content* to start searching for their desired pet, using criteria as filters, which can be done by themselves or by asking the Google Assistant to do so, since *Browse Content* is an Use Case in which both actors are involved. From the results obtained, the user can check the pets’ information and if they are interested in a specific pet, they can *Initiate Adoption process*, by *Filling Adoption Intent form* just asking Google Assistant to activate that Use Case, where they will be able to fill up an Intent Form to be sent to the Shelter.

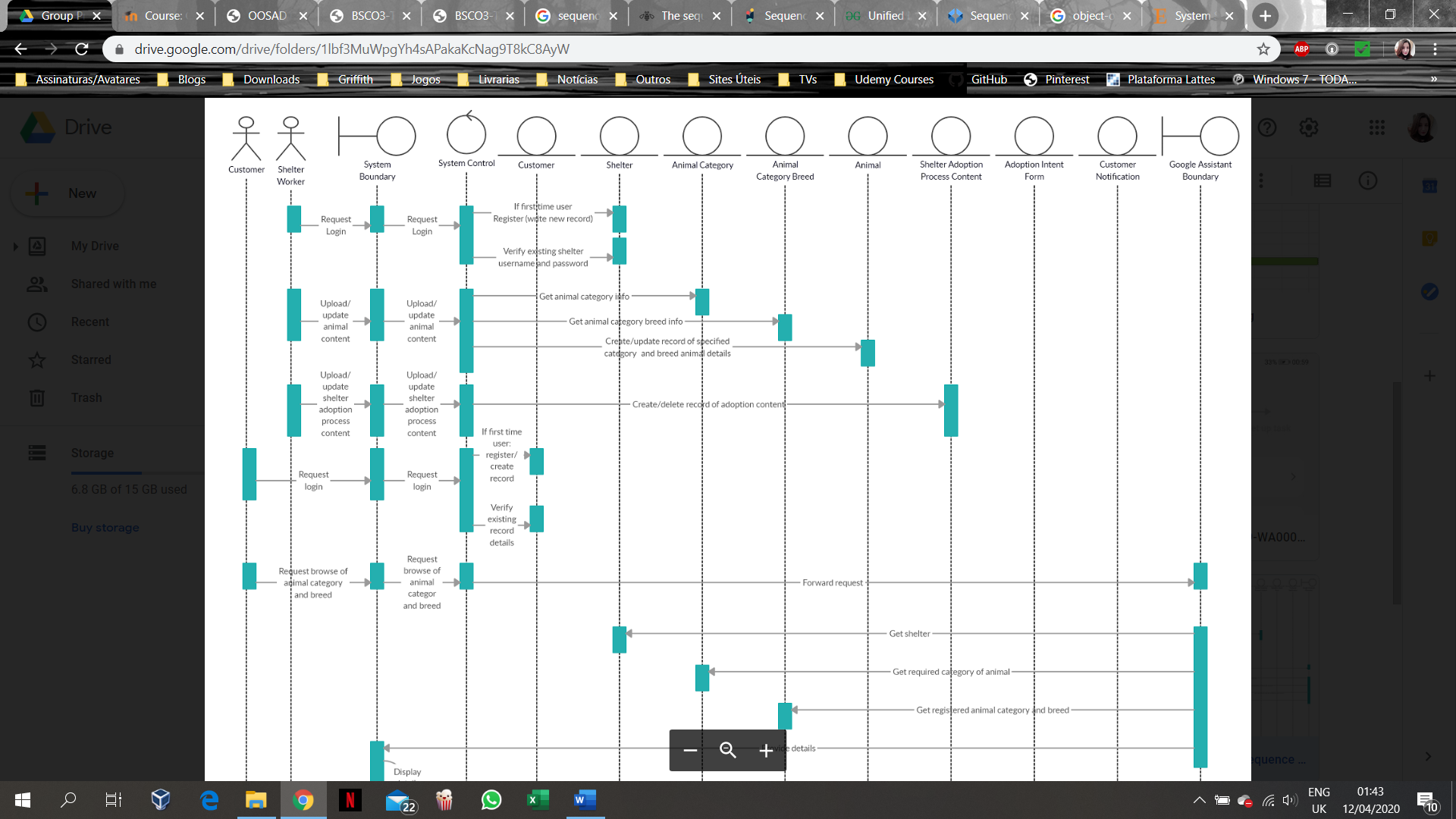
In case of a Shelter Worker, after successful *Login* functionality, they will be able to *Upload/update Animal content*, *Upload/update Adoption process content*, as well as *Browse Received Intent forms for Adoption* from Customers. In case of the latter functionality, they will also be able to *Notify the Customer for Visit of chosen pet*.

Both, the Shelter Worker and the Customer, can in some cases, start a *Chat* since both are involved in the *Chat* Use Case. If they have forgotten their password, or need to update the password, a *Request to* *Reset password* can be sent, in which functionality the Admin is involved.

**3.5.2 Sequence Diagram**

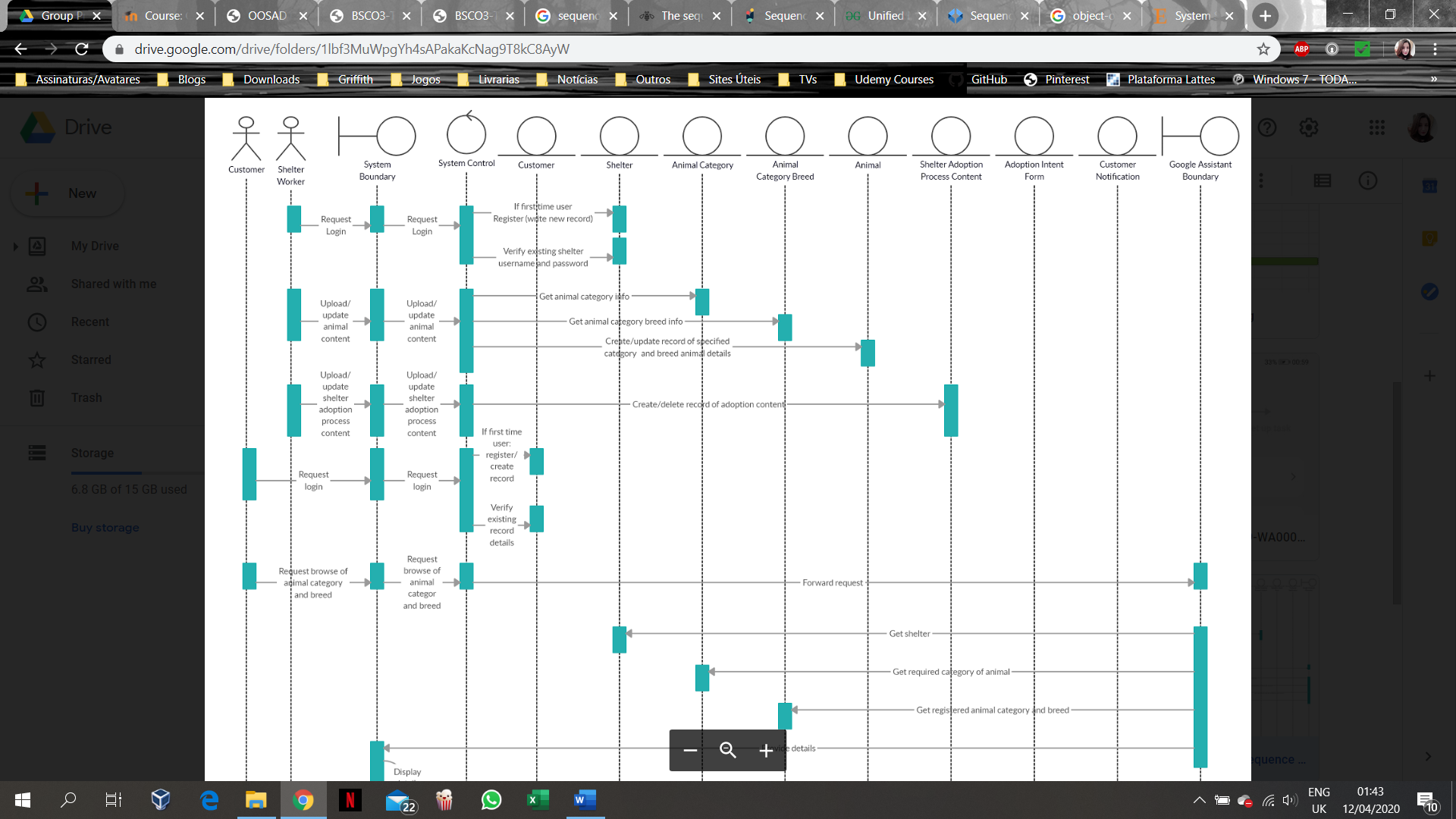
Sequence Diagrams illustrate the interactions between objects within the system, through the messages sent over period of time between them (Smart Draw, 2020). They describe how and in what order the objects’ interaction takes place. The basic components of a Sequence diagram are the actors, the objects with their lifelines, and the messages (Jain, 2019).

The actors in a Sequence diagram are represented with a stick figure notation, same as in the Use Case diagram. In the Sequence diagram, the actor represents the person, another system or system component that plays a role in the studied system, and that will interact with the system’s objects. A Sequence diagram can have one or more actors (Jain, 2019).



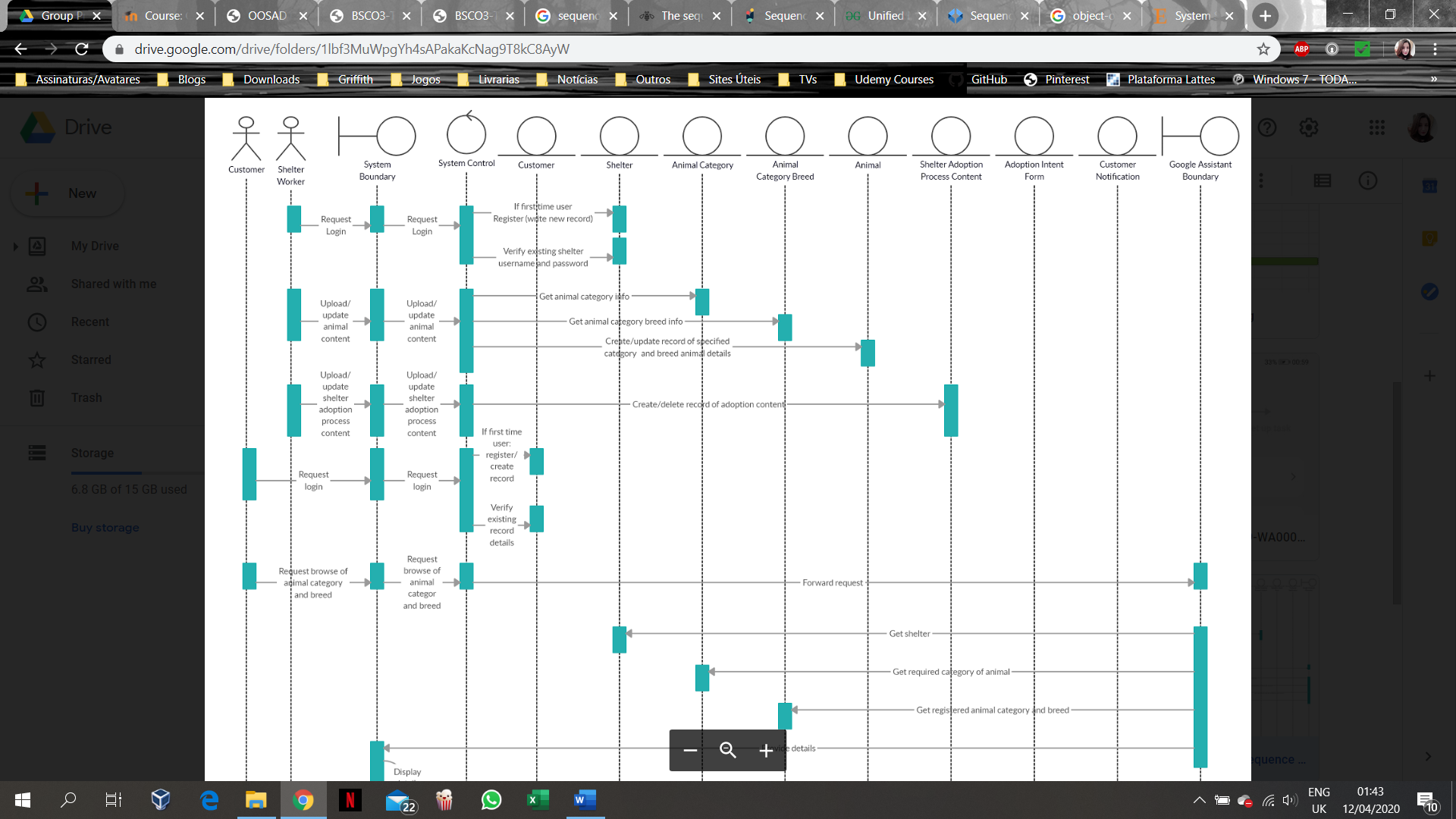
*Figure 3.11: Visual Representation of an Actor in a Sequence Diagram.*

The system objects are presented on the diagram with their vertical lifelines. Thin boxes on the lifelines, called Focus on control, represent the time the interactions take place on the objects’ lifelines in a sequence that they happen. The lifeline of an actor demonstrates when an action with the system is initiated by the actor (Athuraliya, 2017). All object classes in the system (the Boundary class, the Control class, and Persistent Entity classes) are presented with their lifelines.



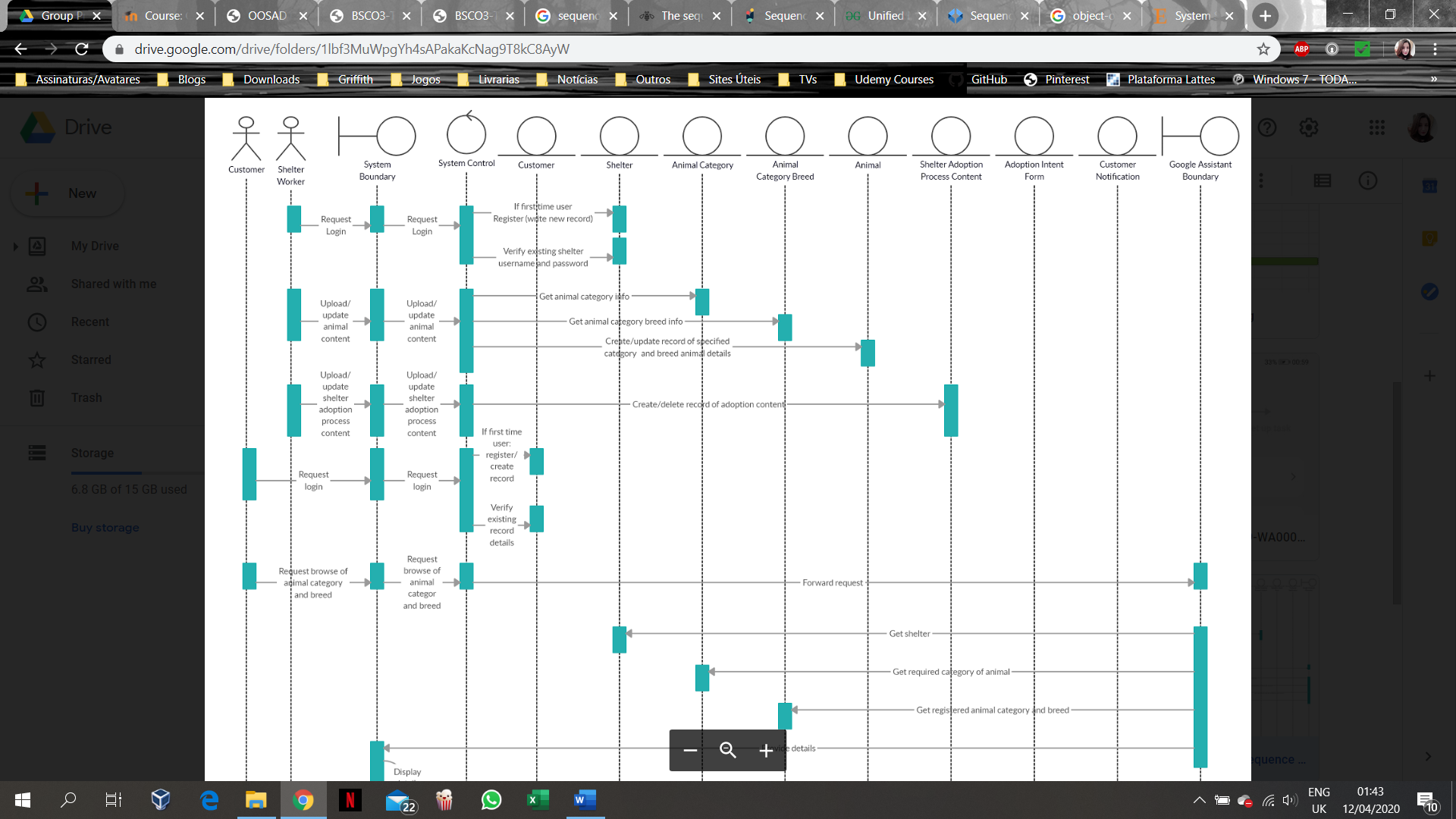
*Figure 3.12: Visual Representation of the Objects Lifelines in a Sequence Diagram.*

The Boundary class is responsible for receiving the Actor’s messages and displaying any results of the internal processing to the respective Actor. If the message is not the Boundary’s responsibility, the Boundary class is forwarding the message to the System Control.



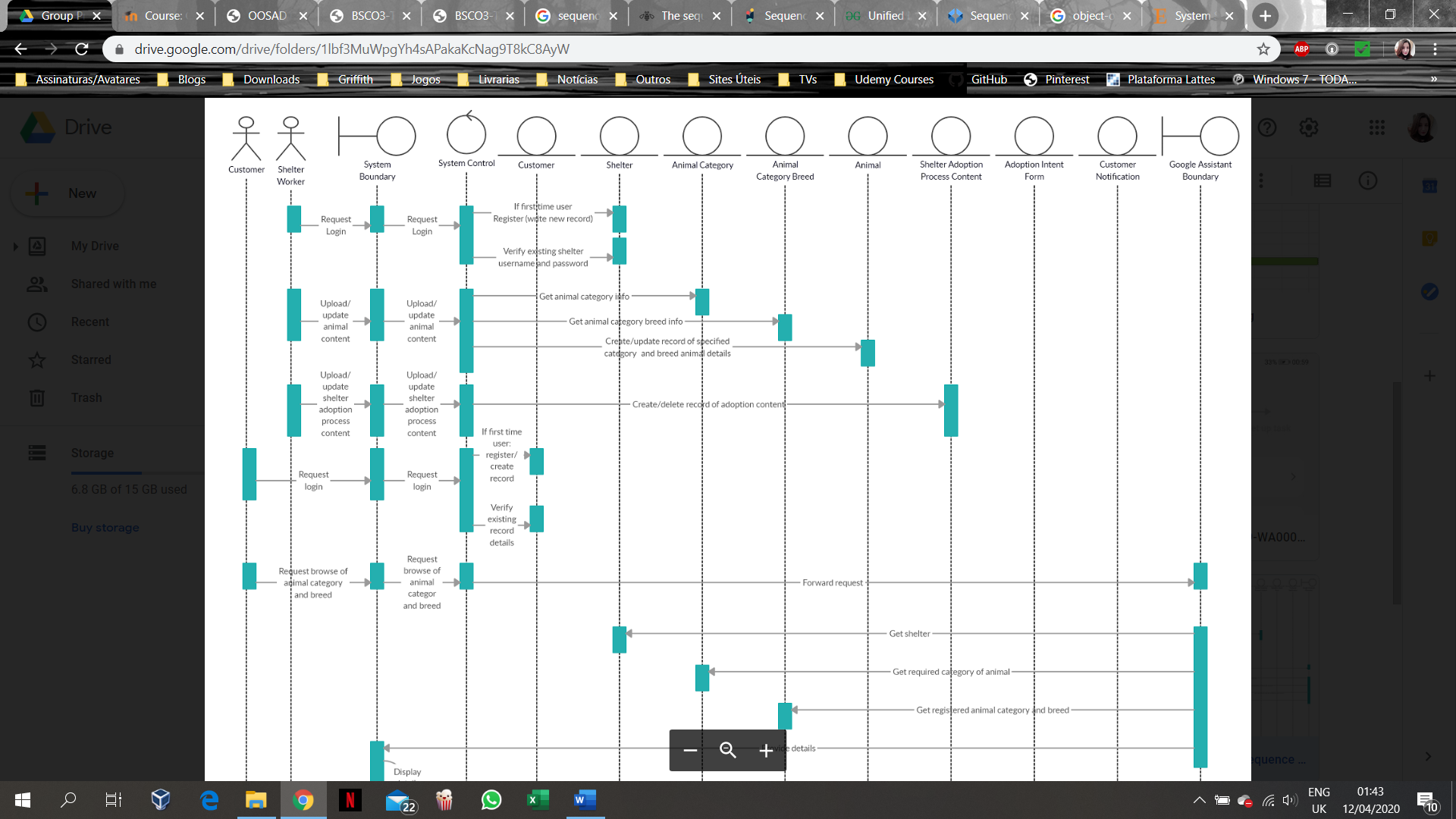
*Figure 3.13: Visual Representation of the System Boundary in a Sequence Diagram.*

The System Control class is responsible for managing the system. It determines which Persistent Entity class is the recipient of a message, and forward the message to the class responsible for processing it. It is also responsible for returning any responses that it got from the entity classes to the boundary class, so the response can be displayed to the actor.



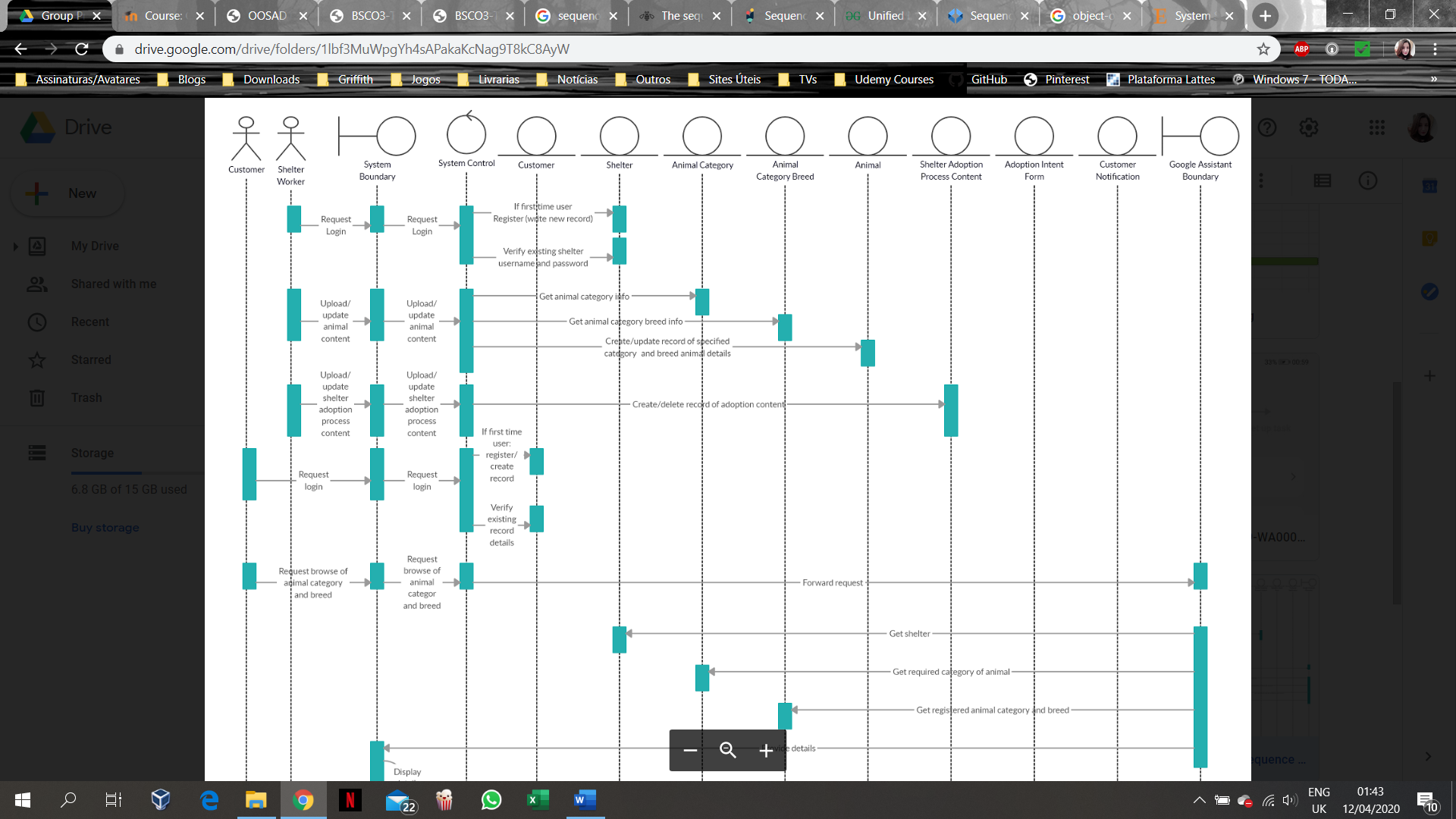
*Figure 3.14: Visual Representation of the System Control in a Sequence Diagram.*

The Entity classes represent the Persistent objects within the system, which at Design and Development stage will become Database Tables of the system and will exist throughout the system’s life. They are the entities that encapsulate specific attributes and methods. The Persistent Entity classes are going to receive Actor’s requests that were forwarded from the System Control class, process those messages and return a result to be displayed on the System Boundary class.



*Figure 3.15: Visual Representation of an Entity Class in a Sequence Diagram.*

The messages in a Sequence diagram are the requests that are going to travel through the system and be processed at some stage. They are placed onto the lifelines and rewritten every time they are forwarded to a different class. When a message is not redirected to another class, but it is forwarded to the same one instead, it is called a self-message. The direction in which a message travels is represented by an arrow pointing to the class’ lifeline that is going to receive the message.



*Figure 3.16: Visual Representation of Messages being sent and forwarded in a Sequence Diagram.*

The Sequence diagram for the current Project is described as follows:

***Actors:***

* Customer
  + Messages:
    - Request Login;
    - Request Browse Animal category and breed;
    - Choose Shelter Animal from the list;
    - Choose to Fill Adoption Intent Form.
* Shelter Worker:
  + Messages:
    - Request Login;
    - Upload/update Animal content;
    - Upload/update Shelter Adoption Process content;
    - Create Notification for Customer to Visit chosen pet.

***Persistent Entities:***

* Customer – holds Customer’s information;
* Shelter – holds Shelter’s information;
* Animal category – holds Animal categories information;
* Animal Breed – holds Animal Breed information;
* Animal – holds specific Animal’s information, e.g. name, age, size, personality;
* Shelter Adoption Process content – used to store information, regarding both Shelter and Customer, as well as to store the link that will lead the user to the actual documentation;
* Adoption Intent Form – used to store information regarding both customer and animal, as well as to store the adoption intent form text;
* Customer notification – Used to store information regarding both shelter and customer, as well as the actual notification information.

The current project presents two different boundaries. One is the System Boundary that is going to receive and redirect the Actor’s requests to the System Control, and the other is the Google Assistant’s Boundary. The Google Assistant boundary is going to receive some of the forwarded requests from the System Control, retrieve the necessary information from the entity classes and then forward the results to the System boundary, so it can be displayed to the actors.

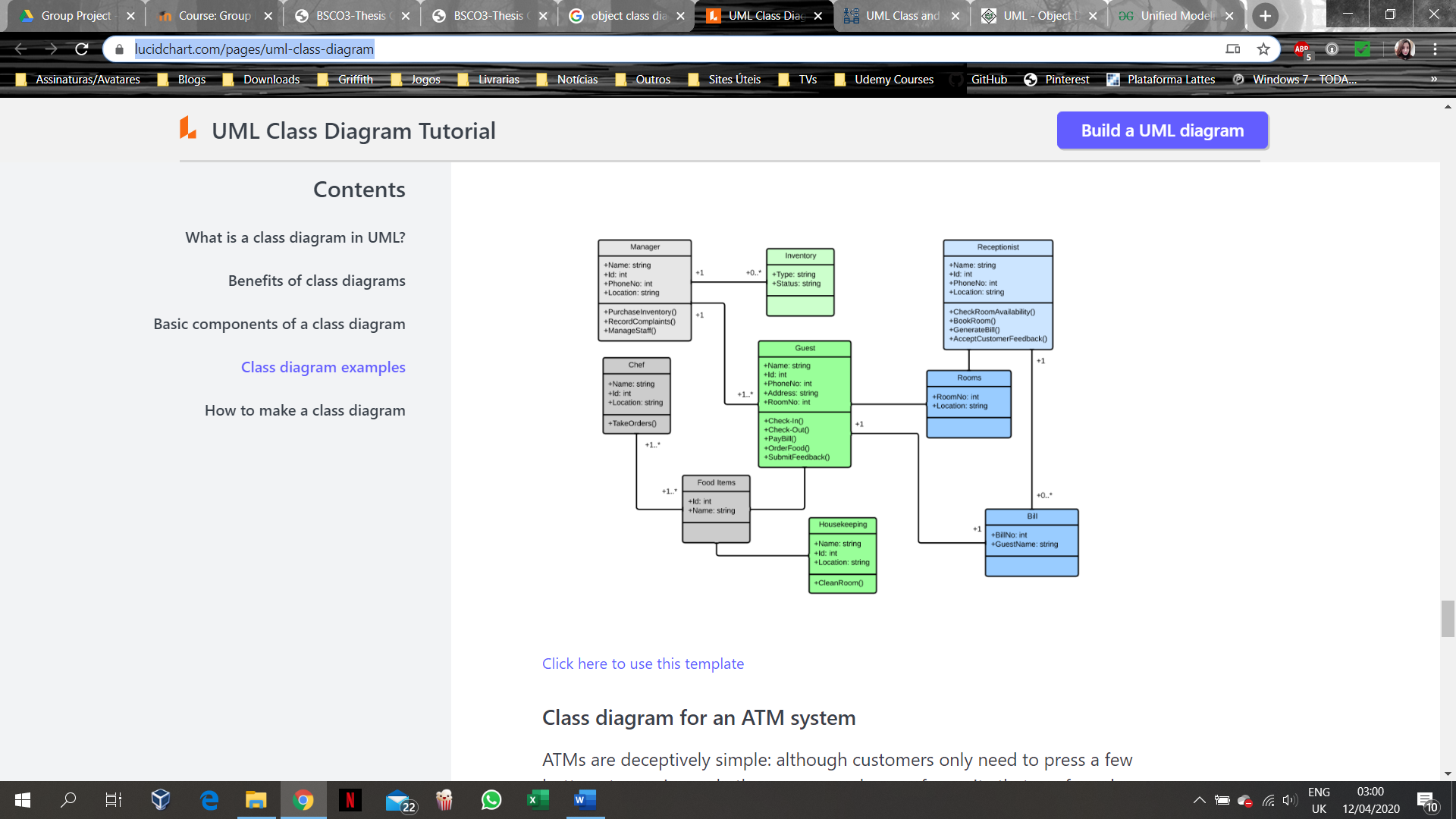
The full representation of the Sequence diagram for the current project is presented in *Appendix A6*.

**3.5.3 Object Class Diagram**

An Object Class Diagram presents the objects within the system and the nature of their relationships. They allow the developer to visualise the structure of the system at a given time and to understand the functional requirements of the system (Jain, 2020).

The visual representation of an object in an Object Class diagram is a rectangle with three different sections. The top one is reserved for the name of the object class, the middle one is used to display all the object class’ attributes, while the bottom one is used to store the methods that operate on the attribute values of this object class. The attributes in the middle section are preceded with a minus sign (-), which indicates a private information that can only be accessed by the class itself, while the methods at the bottom section are preceded by a plus sign (+) (Lucid Chart, 2020b).

The lines connecting object classes on the diagram represent the relationships between them. The number placed close to a class indicates the cardinality of the relationship with the class on the other end of the relationship. The most common cardinalities of relationships are: zero to one, one to one, zero to many, one to many and many to many (Jain, 2020).



*Figure 3.17: Example of Visual Representation of an Object Class Diagram (Lucid Chart, 2020b).*

The Object Class diagram for the current Project is described as follows:

***Animal***

* Attributes:
* Animal ID (P.K.);
* Category ID (F.K.);
* Breed ID (F.K.);
* Shelter ID (F.K.);
* Name;
* Gender;
* Size;
* Age;
* Chip number;
* Picture;
* Personality.
* Methods:

+ Create record;

+ Update record;

+ Get chosen animal details.

***Animal category***

* Attributes:
* Category ID (P.K.);
* Category name.
* Methods:

+ Get animal category information.

***Animal category breed***

* Attributes:
* Breed ID (P.K.);
* Category ID (F.K.);
* Breed name.
* Methods:

+ Get animal category breed information.

***Shelter***

* Attributes:
* Shelter ID (P.K.)
* Shelter name;
* Shelter address;
* Shelter phone number;
* Shelter username (e-mail);
* Shelter password.
* Methods:

+ Create New Record;

+ Verify login details;

+ Get shelter information.

***Shelter adoption process content***

* Attributes:
* Adoption document ID (P.K.);
* Shelter ID (F.K.);
* Document link.
* Methods:

+ Create new document record;

+ Delete existing record.

***Customer***

* Attributes:
* Customer ID (P.K.);
* Name;
* Age;
* Phone number;

- Customer username (e-mail);

- Customer password.

* Methods:

+ Create New Record;

+ Verify login details.

***Adoption Intent Form***

* Attributes:
* Form ID (P.K.);
* Animal ID (F.K.);
* Customer ID (F.K.);
* Adoption Intent Form text.
* Methods:

+ Create new record.

***Notification***

* Attributes:
* Notification ID (P.K.);
* Shelter ID (F.K.);
* Customer ID (F.K.);
* Notification information.
* Methods:

+ Create new record.

The full representation of the Object Class diagram for the current Project is presented in the *Appendix A7*.

## 3.6 Alternative Design Solutions

Three different Design solutions were analysed for the development of the current project, taking into consideration its time constrained nature, where each of the alternatives presents different level of functionalities that could be implemented in each one of them.

### 3.6.1 Low-End Alternative

The Low-End alternative presents the basic functionalities of the system, without which the application cannot exists. This alternative involves as actors the customer, the shelter worker and the admin, only. The basic functionalities are as follows:

* Login / Registration;
* Reset password;
* Upload/update content (animal content, adoption process content);
* Browse content.

### 3.6.2 Mid-Range Alternative

The Mid-Range alternative represents compromise between both, the Low-End and the High-End alternatives, and all actors are involved in this solution. Therefore, the Mid-Range alternative includes **all** functionalities of the Low-End solution, **plus**:

* Initiate Adoption process;
* Fill Adoption Intent Form (within Initiate Adoption process);
* Notify Customer for Visit.

### 3.6.3 High-End Alternative

The High-End alternative not only includes all functional requirements for the system, but also provides extra features that the users may wish for. This alternative also involves all actors in the system. Therefore, the High-End alternative includes **all** the functionalities of the Low-End and Mid-Range solutions, **plus**:

* Chat with Shelter Worker

## 3.7 Alternative to be implemented

As discussed at the beginning of this chapter, the Android Operating System surpasses the iOS market share in Ireland, and even though it is just a slight difference of 11%, it is the most widely used mobile operating system in the world. Based on that and the fact that the members of this project have a previous experience with Java language and that the Android Operating System can be developed using Java, the current project will be based on an application using Android Operating System. Another reason for choosing Android Operating System for development of the application is because of the integration of the application with the smart home device since the *Google Nest Hub* is compatible with *Android* applications.

As already mentioned, due to time constraints for the project’s development and the skills level of the developers, the Dynamic Software Development Methodology was the one chosen to be implemented in this project. The Alternative Design Solution to be used in the Development stage will be the Mid-Range solution, due to limited timeframe of the project.

# 3.8 Chapter Summary

This chapter has focused on research of different mobile operating systems, smart home devices, Object-Oriented programming characteristics, Software Development Methodologies, types of Unified Modelling Language diagrams and Alternative Design solutions.

From the research, discussed in this chapter, it was possible to establish that the Android Operating System and the iOS are the mobile operating systems that are most used worldwide, where the *Android* leads the market in Ireland, closely followed by the iOS. Also, it was possible to determine that in order to create a robust application, it is necessary to include the Object-Oriented Programming concepts and specify the most appropriate methodology to be used, while developing it. The used Unified Modelling Language diagrams for the Project’s Conceptual model gave the developers a better view of the requirements for the project, as well as made it possible to design different scenarios that can be accomplished according to the project’s constraints, varying from the simplest form of application that is going to satisfy the project’s minimum requirements to an alternative that presents a high-level of functionalities.

The next chapter will describe the developing process of the current project, from the design to the actual coding of the project’s application. It will show in details step-by-step the application’s building process.

# REFERENCES

Abellán, E. (2020). *WAM.* [Online]   
Available at: https://www.wearemarketing.com/blog/what-is-the-agile-methodology-and-what-benefits-does-it-have-for-your-company.html  
[Accessed 11 April 2020].

Agile Modeling (2020). *UML 2 - Use Case Diagramming Guidelines.* [Online]   
Available at: http://www.agilemodeling.com/style/useCaseDiagram.htm  
[Accessed 09 April 2020].

Anderson, N., McGowan, A., Hanna, P. & Busch, J. (2015). Strange, but true? Object-Oriented programming is best taught, and learnt, while sitting on the floor. *Australian Association for Research in Education*, 29 November, p. 15. [Online]

Available at: https://files.eric.ed.gov/fulltext/ED593835.pdf

[Accessed 07 April 2020].

Appelo, J. (2008). *The Definitive List of Software Development Methodologies.* [Online]   
Available at: https://noop.nl/2008/07/the-definitive-list-of-software-development-methodologies.html  
[Accessed 11 April 2020].

Athuraliya, A. (2017). *Sequence Diagram Tutorial: Complete Guide with Examples.* [Online]   
Available at: https://creately.com/blog/diagrams/sequence-diagram-tutorial/  
[Accessed 10 April 2020].

Callaham, J. (2019). *The history of Android OS: its name, origin and more.* [Online]   
Available at: https://www.androidauthority.com/history-android-os-name-789433/  
[Accessed 07 April 2020].

GeeksforGeeks (2020). *Kotlin Programming Language.* [Online]   
Available at: https://www.geeksforgeeks.org/kotlin-programming-language/  
[Accessed 08 April 2020].

Guru99 (2020). *Dynamic Software Development Method.* [Online]   
Available at: https://www.guru99.com/agile-scrum-extreme-testing.html#10  
[Accessed 11 April 2020].

IntellectSoft (2019). *Top 12 Software Development Methodologies You Should Know.* [Online] Available at: https://www.intellectsoft.net/blog/top-12-software-development-methodologies-you-should-know/  
[Accessed 08 April 2020].

Islam, M. (2017). *RAD.* [Online]   
Available at: http://sddmohammedislam.weebly.com/rad.html  
[Accessed 11 April 2020].

Jain, A. (2019). *Unified Modeling Language (UML) - Sequence Diagrams.* [Online]   
Available at: https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/  
[Accessed 10 April 2020].

Jain, A. (2020). *Unified Modeling Language (UML) - Object Diagrams.* [Online]   
Available at: https://www.geeksforgeeks.org/unified-modeling-language-uml-object-diagrams/  
[Accessed 10 April 2020].

JavaTPoint (2018). *Waterfall Model.* [Online]   
Available at: https://www.javatpoint.com/jira-waterfall-model  
[Accessed 10 April 2020].

Lucid Chart (2020a). *UML Use Case Diagram Tutorial.* [Online]   
Available at: https://www.lucidchart.com/pages/uml-use-case-diagram  
[Accessed 09 April 2020].

Lucid Chart (2020b). *UML Class Diagram Tutorial.* [Online]   
Available at: https://www.lucidchart.com/pages/uml-class-diagram  
[Accessed 10 April 2020].

LucidChart Content Team (2019). *4 Phases of Rapid Application DevelopmentMethodoly.* [Online]   
Available at: https://www.lucidchart.com/blog/rapid-application-development-methodology  
[Accessed 11 April 2020].

Monus, A. (2020). *6 OOP Concepts in Java with examples.* [Online]   
Available at: https://raygun.com/blog/oop-concepts-java/#association  
[Accessed 08 April 2020].

Muslihat, D. (2018). *Agile Methodology: An Overview.* [Online]   
Available at: https://zenkit.com/en/blog/agile-methodology-an-overview/  
[Accessed 11 April 2020].

Nakov, S., Dimitrov, D., Germanov, H., Murdanliev, I., Stoynov, M., Bivas, M., Kostov, N., Nedyalkov, N., Vasilev, N., Donchev, P., Hadjieva, P., Ivanov, R., Kirilov, R., Todorov, R., Zlatinov, S., Staev, S., Nakov, S., Bozhikov, T., Stoev, T., Konov, T., Georgiev, V., Kolev, V., Pavlov, V., Yosifov, Y. (2013). *Fundamentals of computer programming with C#. The Bulgarian C# book.* 1 ed. Bulgary: Faber Publishing.

Available at: https://introprogramming.info/wp-content/uploads/2013/07/Books/CSharpEn/Fundamentals-of-Computer-Programming-with-CSharp-Nakov-eBook-v2013.pdf

[Accessed 08 April 2020].

Pal, S. K. (2019). *Software Engineering - Spiral Model.* [Online]   
Available at: https://www.geeksforgeeks.org/software-engineering-spiral-model/  
[Accessed 11 April 2020].

Project Manager (2020). *Waterfall Methodology in Project Management.* [Online]   
Available at: https://www.projectmanager.com/software/use-cases/waterfall-methodology  
[Accessed 09 April 2020].

Raphael, J. (2020). *Android versions: A living history from 1.0 to 10.* [Online]   
Available at: https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html  
[Accessed 05 April 2020].

Reisinger, D. (2020). *Google Home Vs Amazon Echo: A Comparison.* [Online]   
Available at: https://www.forbes.com/sites/forbes-personal-shopper/2020/02/06/google-home-vs-amazon-echo-a-comparison/  
[Accessed 11 April 2020].

Sacolick, I. (2020). *What is Agile methodology? Modern software development explained.* [Online]   
Available at: https://www.infoworld.com/article/3237508/what-is-agile-methodology-modern-software-development-explained.html  
[Accessed 11 April 2020].

Sharma, R. (2019). *Developing for Android - An Introduction.* [Online]   
Available at: https://www.cprogramming.com/android/android\_getting\_started.html  
[Accessed 08 April 2020].

Silva, N. (2019). *Use Case Diagram Relationships Explained with Examples.* [Online]   
Available at: https://creately.com/blog/diagrams/use-case-diagram-relationships/  
[Accessed 09 April 2020].

Smart Draw (2020). *Sequence Diagram.* [Online]   
Available at: https://www.smartdraw.com/sequence-diagram/  
[Accessed 10 April 2020].

StatCounter (2020a). *Mobile Operating System Market Share Ireland.* [Online]   
Available at: https://gs.statcounter.com/os-market-share/mobile/ireland  
[Accessed 7 April 2020]

StatCounter (2020b). *Mobile Operating System Market Share Worldwide.* [Online]   
Available at: https://gs.statcounter.com/os-market-share/mobile/worldwide  
[Accessed 8 April 2020].

TatvaSoft (2015). *Top12 Software Development Methodologies & its Advantages/Disadvantages.* [Online]   
Available at: https://www.tatvasoft.com/blog/top-12-software-development-methodologies-and-its-advantages-disadvantages/  
[Accessed 11 April 2020].

Tryqa (2017). *What is RAD model - advantages, disadvantages and when to use it?.* [Online]   
Available at: http://tryqa.com/what-is-rad-model-advantages-disadvantages-and-when-to-use-it/  
[Accessed 11 April 2020].

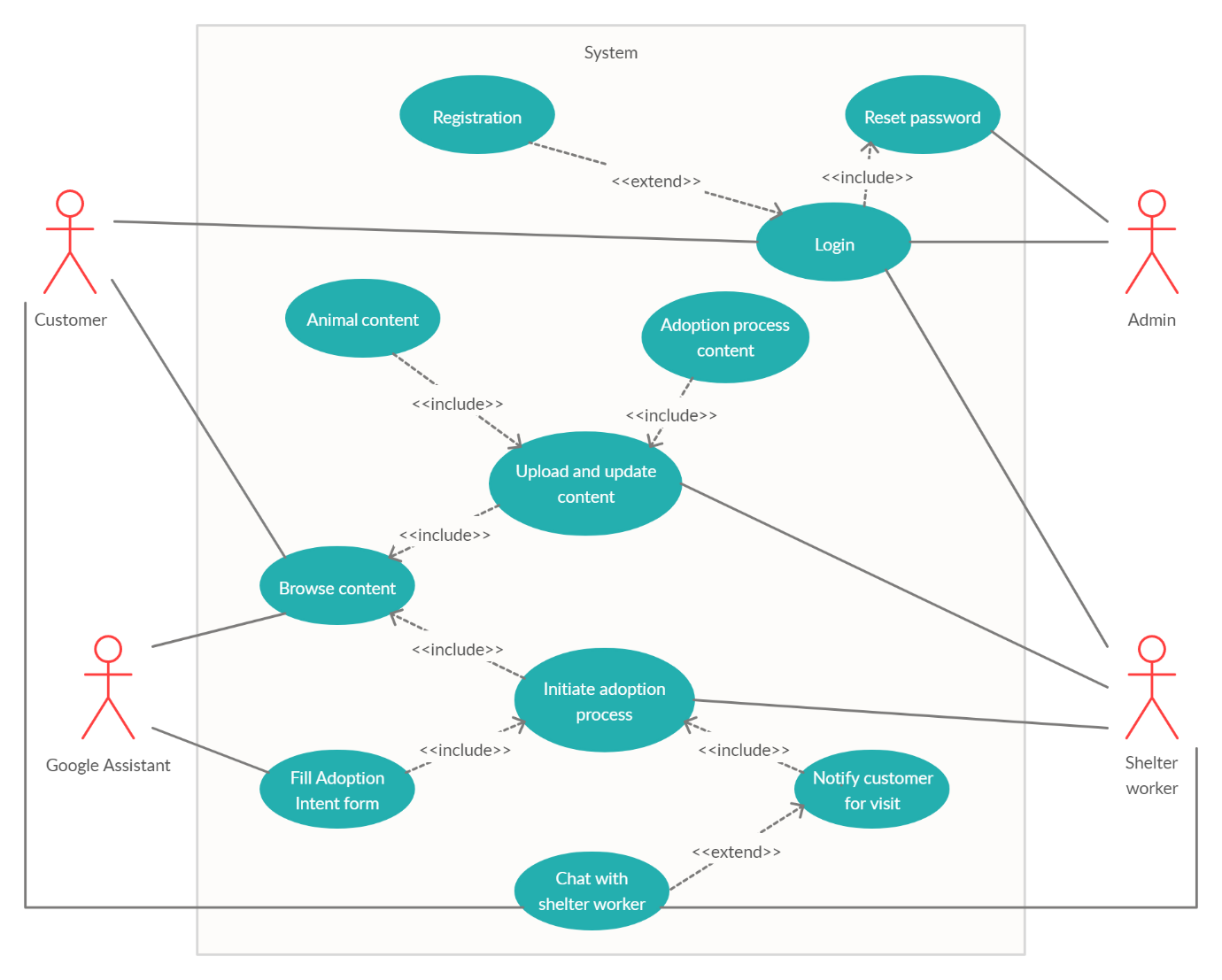
Tutorials Point (2020). *UML - Use Case Diagrams.* [Online]   
Available at: https://www.tutorialspoint.com/uml/uml\_use\_case\_diagram.htm  
[Accessed 09 April 2020].

The Tool (2018). *Differences Between Android and iOS in Mobile App Developmentl.* [Online]   
Available at: https://thetool.io/2018/mobile-app-development-ios-vs-android  
[Accessed 07 April 2020].

Visual Paradigm (2020). *What is Unified Modeling Language (UML)?.* [Online]   
Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/  
[Accessed 09 April 2020].

Viva Differences (2019). *Difference Between Spiral Model and Prototype Model In Software Engineering.* [Online]   
Available at: https://vivadifferences.com/difference-between-spiral-model-and-prototype-model-in-software-engineering/  
[Accessed 11 April 2020].

# APPENDIX A5



# APPENDIX A6

# APPENDIX A7