Chapter 3: Project Application Analysis

From the previous two chapters, Secondary and Primary research were carried out and discussed on the project topic. This has allowed for the proven feasibility and necessity for the development of a mobile application that helps people with arm impairments to read without the use of touch controls. For such development to be possible, an appropriate Development methodology must be implemented before real development begins.

In this chapter, Development methodologies used in today's industry will be described in depth and advantages and disadvantages will be compared in the process. A selection of the most appropriate Development methodology for the project to follow will then be made based on this analysis.

3.1 Mobile Operating Systems Overview

The mobile device has become a primary tool in society's ability to perform essential tasks. There are many different systems on the market that serve their own specific function for the end-user, however, according to StatCounter (gs.statcounter.com), *Android* is leading at 62% of the market share, whereas iOS is down at over 16%, as of December 2020.

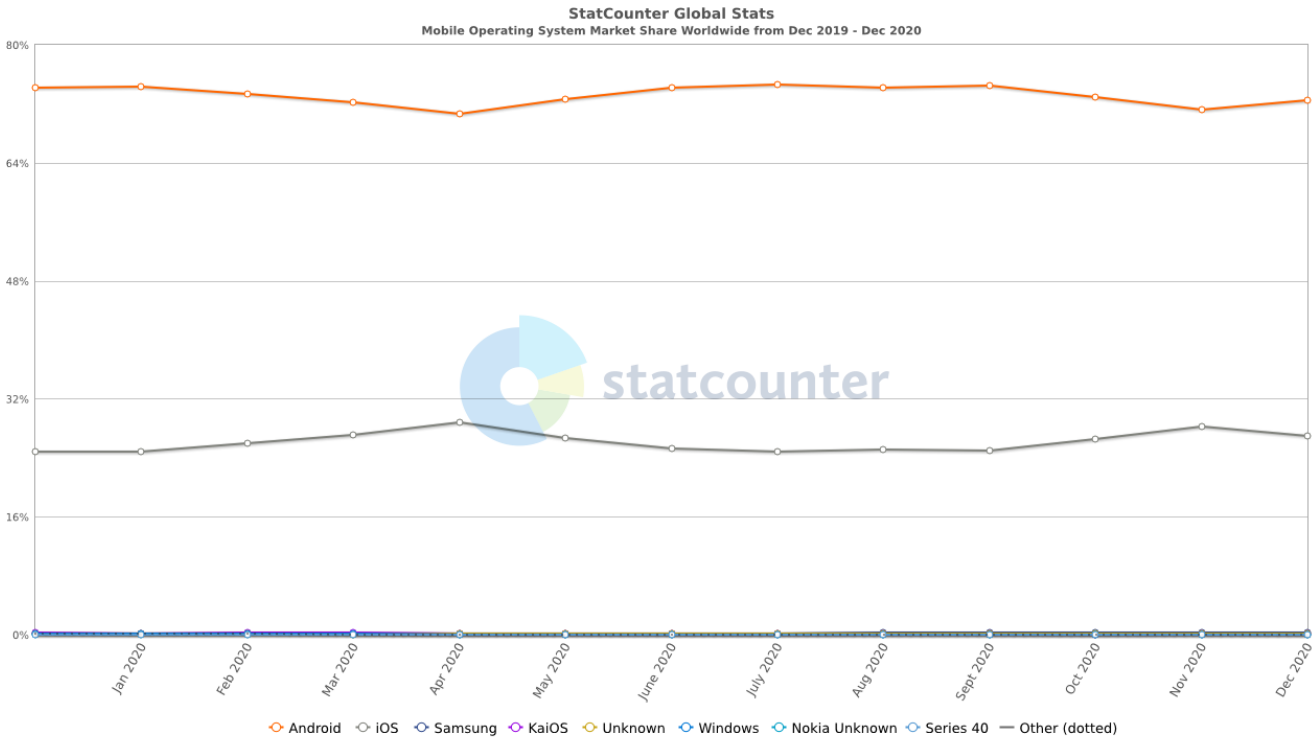


Figure 3.1 - Android vs iOS (gs.statcounter.com)

Due to this fact, it would be ideal to develop for the Android market, as it has the most users (gstatcounter, 2020).
Over the years, Android has come out with new iterations improving on itself from 2008 to 2021 as of this year. The versions are presented as follows:

- 2008 – Version 1.0
- 2008 – Version 1.1
- 2009 – Version 1.5 Cupcake
- 2009 – Version 1.6 Donut
- 2009 – Version 2.0 Eclair
- 2009 – Version 2.1 Eclair
- 2010 – Version 2.2 Froyo
- 2010 – Version 2.3 Gingerbread
- 2011 – Version 3.0 Honeycomb
- 2011 – Version 3.2 Honeycomb
- 2011 – Version 4.0 Ice Cream Sandwich
- 2012 – Version 4.1 Jellybean
- 2013 – Version 4.3 Jellybean
- 2013 – Version 4.4 Kitkat
- 2014 – Version 5.0 Lollipop
- 2014 – Version 5.1 Lollipop
- 2015 – Version 6.0 Marshmallow
- 2016 – Version 7.0 Nougat
- 2017 – Version 8.0 Oreo
- 2018 – Version 9.0 Pie
- 2019 – Version 10
- 2020 – Version 11

(Android, 2021)
The programming languages that are most used in development of an Android application are Java and Kotlin. Java is known as the official language for Android development and is supported with the main Integrated Development Environment, *Android Studio*. Java is more versatile and has other uses outside of Android, unlike Kotlin, which is supposed to be easier to understand (Sinickl, 2019).
As the current project is to be developed with the official IDE *Android Studio* for better integration of the application, the Java programming language will be utilised due to the developer being more familiar with this environment.

3.1.1 Voice Assistants Overview

With the rise of Voice Assistants in our homes, it is necessary to analyse what has been created throughout the past couple of years. Most notable is the Amazon's *Alexa* which seems to have

captured the market early on with their smart home device *Echo Dot*. The issue with Amazon's *Alexa* is that it is tied to Amazon's ecosystem, which means it has limited functionality for developing and integrating it with Google's *Android* environment. Apple's *Siri* cannot be developed for, as *Siri* is strictly for use in *Apple* products only, and not *Android*.
Because of these limitations, the current project application to be developed will be integrated the with the *Google Assistant*, which has its own open API, documentation and guides to help develop for (T4, 2020).

3.2 Object Oriented Programming

Object Oriented Programming (OOP) is characterised with object instances and classes. A *class* contains information and instructions that, when run, perform certain behaviours. Classes interact with other parts of the environment. There are multiple basic principles of Object-Oriented Programming: Data is given higher priority than functions; objects communicate through member functions; data cannot move out of the object; and a bottom-up design approach is adopted. Objects are the basic run-time entities in OOP (Somashekara, *et al*., 2017). The features of Object-Oriented environment are as follows:

- **Encapsulation**: In an OOP program data is hidden and cannot be accessed by any external function. Wrapping data and functions into a class is known as *data encapsulation*. This is a way of achieving data security in an Object-Oriented program. It is possible to prevent modification of data by an external non-member function of a class. Only the functions inside the class will have access to this data (Somashekara, *et al*., 2017).
- **Abstraction**: One of the goals of Object-Oriented Programming is to reduce the complexity by not including background details. Abstraction allows the user to use an object without knowing its internal working. A language permits only functional instruction, which allows the programmer to use a function without knowing the details (Somashekara, *et al*., 2017).
- **Inheritance**: A new class can be constructed from an existing class without affecting that currently existing class. The new class will have the same features as the higher level class has, from which they have been derived, and in addition will have its own features. The existing class is known as the *base class* and the new class, derived from this class, is known as a *derived class*. Through *inheritance* it is possible to use the

details of an existing class without repeating the details of base classes which has the advantage of *reusability* and saves development time (Somashekara, *et al*., 2017).

- **Polymorphism**: This is another feature of OOP, in which a function can take many forms, based on its type of arguments, number of arguments and an operator can take different behaviours, depending on operand of the corresponding operator. When an operator behaves differently, based on the operand, it is said the operator is *overloaded*. Likewise, the function is said to be *overloaded* when the same function name is used for multiple tasks in the same program (Somashekara, *et al*., 2017).
- **Aggregation**: This uses a "Has a" relationship between two classes. The relationship is only one way, and only one class is dependent on the other (Somashekara, *et al*., 2017).
- **Composition**: This is when two different classes are mutually dependent on each other, in where neither can exist without the other (Somashekara, *et al*., 2017).

3.2.1 Advantages of OOP

- *Security:* Data is encapsulated along with the methods inside a class. Data is safe from modification, which prevents tampering of the program.
- *Reusability*: Because of inheritance, it is possible to use features of an already existing class, without recreating the code base again, which allows for faster development time and increased productivity.
- *Effective communication*: Objects can communicate through passing messages, which makes interfacing without outside systems simple.
- *Easy upgrading*: It is easy to scale the system from small to large using a bottom-up approach.
- *Composing complex work*: It is easy to compose complex work, based on objects, rather than just functions (Somashekara, *et al*., 2017).

3.3 Software Development Methodologies

There are different ways to manage a project. The choice of a suitable Software Development Methodology to manage a project allows for better efficiency. There are methodologies with varying advantages and disadvantages, according to the needs of the project. This presents some of these methodologies and compares the strengths they have over others.

3.3.1 Waterfall Method

The *Waterfall* method was the most used model of development in the early days of software development. It was first realised in 1970 by Winston W. Royce. This method uses a very strict linear approach to development where the stakeholder and customer requirements are gathered at the very beginning of the project, then a sequential project plan is created to accommodate those requirements. Each phase of the project cascades into the next, which is where the term *waterfall* comes from (Gomaa, 2011).

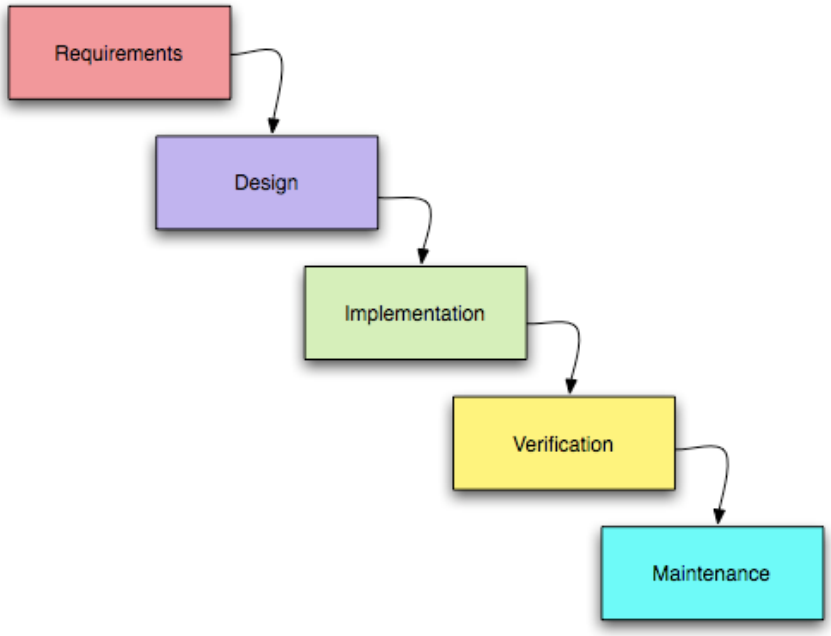


Figure 3 - Waterfall Model (eternalsunshineoftheismind.wordpress.com, 2013)

Advantages:

- Improvement over undisciplined approach, used in early projects
- Simple to implement with clear scheduling
- Good accuracy of estimating development costs, resources, and time on the project
- Project has no delays as requirements are already set in stone

Disadvantages:

- A working system only becomes available late in the life cycle, which means a major design problem will go undetected until it is too late to take an action.
- Entire system must be scrapped if the outcome is incorrect, which wastes a lot of time and money for stakeholders.
- Is not flexible in the case of unexpected events.

- Customers might have an issue with properly defining the requirements of the project at the very start.

(Gomaa, 2011)

3.3.2 Spiral Model

The *Spiral* model is a risk-driven process model, developed by Boehm in 1988. The Spiral model encompasses other life cycle models, like the Waterfall model, the Incremental development model, and the throwaway Prototyping model. In this model, the radial coordinate represents *the cost*, the angular coordinate represents *the progress in the completion* of a cycle of the model.

There are four quadrants, shown on Figure 3.2: Objectives determination and identifying alternative solutions, which gives detailed planning for this cycle; Identify and resolve risks, which gives a detailed assessment of current project risks and plan activities to be performed to resolve these risks; Develop next version of the product; and Review and plan for the next phase/cycle, which is assessing the progress made on the current cycle and preparing for the next one (Gomaa, 2011).

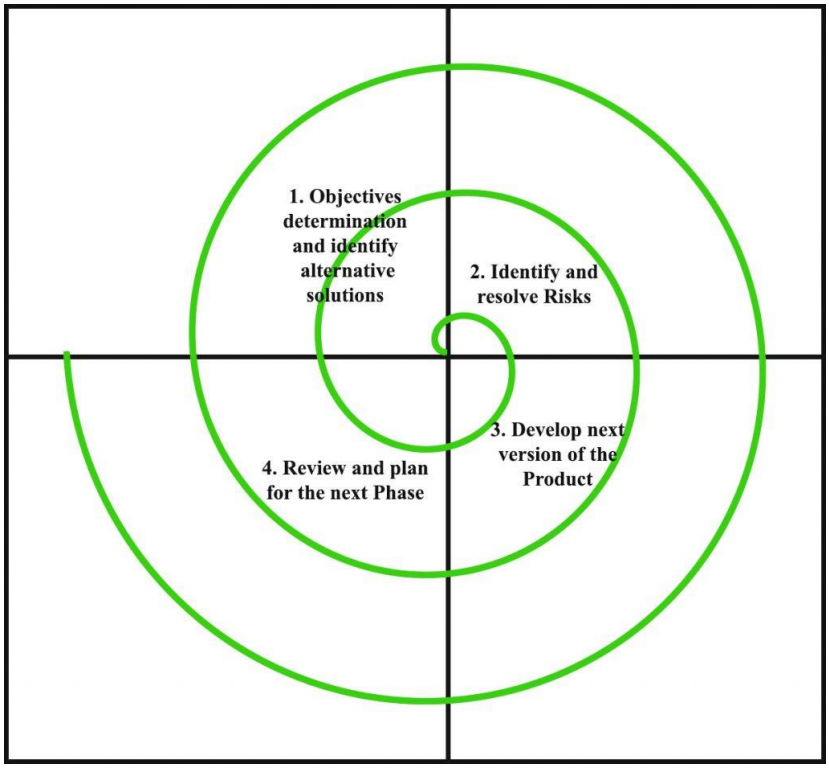


Figure 3.4 - Spiral Model (geeksforgeeks.org, 2021)

Advantages:

- Changes can be done at a later stage.

- The cost estimation is easy, as the prototype building is done in small fragments.
- There is a more robust evaluation of risk.
- There is faster development and features are added in a systematic way.
- Customer feedback is possible in every phase of the cycle.

Disadvantages:

- There is a risk of not meeting development deadline, or budget.
- Only works best for large projects and demands risk assessment expertise.
- Needs documentation, as it has intermediate phases.
- Is not advisable to be used for small projects as ait might cost more than necessary (Guru99, 2021a).

3.3.3 Rapid Application Development (RAD)

Rapid Application Development method focuses on developing rapidly through many iterations and continuous feedback from users. This methodology was introduced by James Martin in 1991, which was a precursor to Agile project management. This methodology reduces the planning phase and maximising the development of the prototype phase. There are 4 phases in the development: Project requirements, Prototype, Rapid construction & feedback gathering, and Finalise product.

For phase 1, developers define a loose set of project requirements by communicating with clients to determine strategies for tackling potential issues that might arise during development. For phase 2, teams start building out the initial models and prototypes. They rapidly produce a working design that can be demonstrated to the client, which encourages user involvement and testing, which results in important feedback. In phase 3, rapid construction begins where application coding, system testing and unit integration occurs, converting prototype and beta systems into a working model. The final phase of RAD is where developers communicate the technical debt gathered in early prototyping, optimising implementation to improve stability and maintainability, as the product is finalised for launch (O'Carroll, 2020).

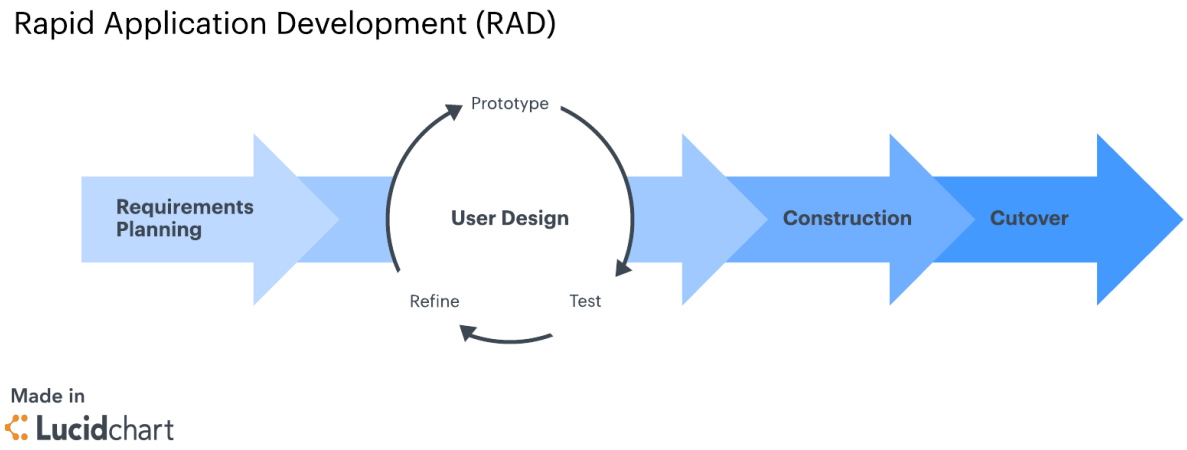


Figure 3.5 – Rapid Application Development (Lucidchart.com, 2021)

Advantages:

- Early system integration & risk reduction
- Adaptability and compartmentalisation of system components
- Iterative releases & faster time to market
- Constant user feedback
- Components are reusable

Disadvantages:

- Relies on a technically strong team for modelling and prototyping
- Difficult for large-scale projects due to requiring modular systems
- Requires high commitment from every stakeholder

(O'Carroll, 2020)

3.3.4 Agile

The *Agile* methodology promotes continuous iteration of development and testing throughout the software development life cycle of the project. It is a term, used to describe software development approaches that employ continual planning, learning, improvement and team collaboration. The *Agile* process involves SCRUM, which is a development method than concentrates on how to manage tasks within a team-based development environment. There are three roles in Scrum: *Scrum master*, who is responsible for setting up the team and sprint meetings, *Product owner* who creates product backlogs and is responsible for delivering the functionality of each iteration, *Scrum team* that manages its own work and organises the work to complete a "sprint", or cycle (Ambler, 2002).

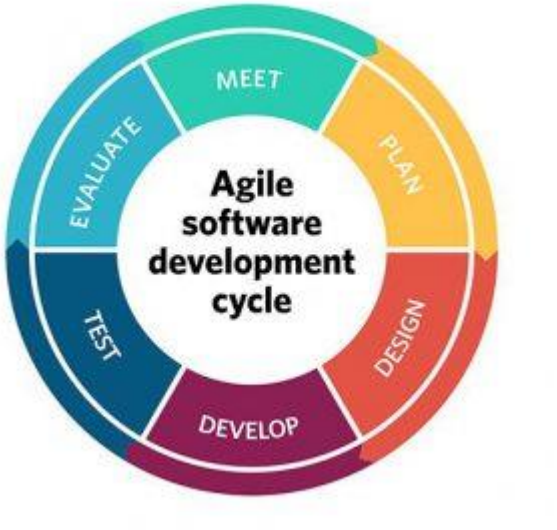


Figure 3.6 - Agile development (Santos, 2021)

- Each iteration of a scrum is known as a *sprint*
- Product backlog is a list, where all details are entered to get the end-product
- During each *sprint*, top user stories are selected and turned into a *Sprint backlog*
- The team checks for a daily *backlog*
- At the end of the *sprint*, the team delivers the product functionality

Advantages:

- The Agile method allows incremental and iterative approach to software design
- Broken into individual models that designers work on
- Customer has early and frequent opportunities to analyse product and make changes to the project
- Small projects are implemented very quickly
- Errors are possible to be fixed anywhere in the project
- Documentation has less priority than the software development

Disadvantages:

- Large projects are difficult to estimate development time
- Agile requires a lot of training and developed skills
- It is not scalable (Guru, 2021b).

3.3.5 Dynamic Software Development Model

The *Dynamic Software Development Method* is similar with the *Agile* approach for software development, where users are actively involved, and the decision-making process is made by the team. This methodology uses new techniques, like Time Boxing (Product Plan, 2021).

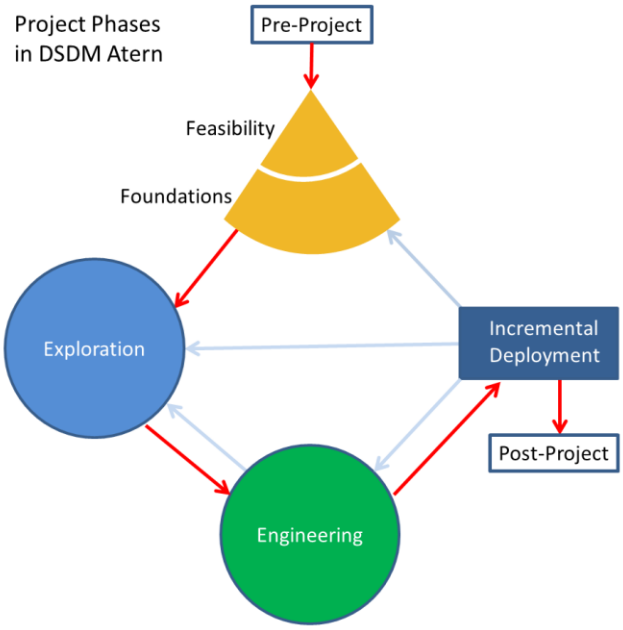


Figure 3.7 - Dynamic Software Development Model (Wikiwand, 2021)

Advantages:

- Product functionality is delivered rapidly
- Developers have fast and easy access to customers
- Projects are completed reliably on time

Disadvantages:

- Can represent big changes in company culture
- Is costly to implement
- Is not ideal for small projects

(Product Plan, 2021)

3.4 Unified Modelling Language

The Unified Modelling Language (UML) is used to specify and visualise the objects of the software systems. It was developed to help software developers communicate with the use of

diagrams what is needed for a program. It is used to get an outside view of the system and what it takes to complete it (Gomaa, 2011).

3.4.1 Use Case Diagram

Use Case diagrams are used to describe a set of actions that systems should perform in collaboration with external users, or Actors of the system. It is used to present an outside view of the inner workings of this system, and how different subjects interact with each other (Fakhroutdinov, 2021).

An Actor specifies the role played by an external entity that interacts with the system. It can be a human user of the designed system or some other third-party system using the services of this subject. The actor must have a name according to the assumed role e.g. Customer, student, passenger. It is shown as a stick figure on the very left or right side of the diagram (Fakhroutdinov, 2021).

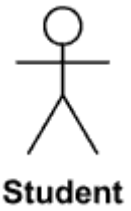


Figure 3.8 - Example of an Actor in a Use Case Diagram

A Use Case is a classifier that specifies a unit of functionality performed by multiple subjects, in which the Use Case applies in collaboration with one or more actors and presents the specified result that is of value to these actors. It is presented as an ellipse that is connected to another Use Case or an Actor (Fakhroutdinov, 2021).

Figure 3.9 - Example of a Use Case in a Use Case Diagram

Each Use Case is a representation of a functionality in which actor(s) are involved. An association, or relationship between an Actor and Use Case indicates that the Actor and the Use Case interact with each other. An Actor may have interactions with several Use Cases and a Use Case may have one or several associated Actors.

There are two relationships between Use Cases that are possible in a Use Case diagram: <<extend>> and <<include>>. *Extend* is a directed relationship that defines an optional

behaviour that is not necessarily meaningful by itself. The *extend* relationship is owned by the Use Case extending from it. <<Extend>> is represented as a dashed line with an arrow directed from the extending (specific) Use Case to the extended (general) Use Case.

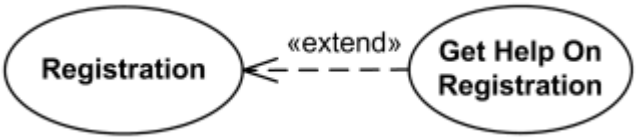


Figure 3.10 - Example of an <<extend>> relationship in a Use Case Diagram

The <<include>> relationship is used to show that the behaviour of an included (general) Use Case is inserted into the function of the including (specific) Use Case. It is used to factor just once on the diagram a common (general) Use Case functionality that might be utilised by several other (specific) Use Cases, while they execute themselves (Fakhroutdinov, 2021).

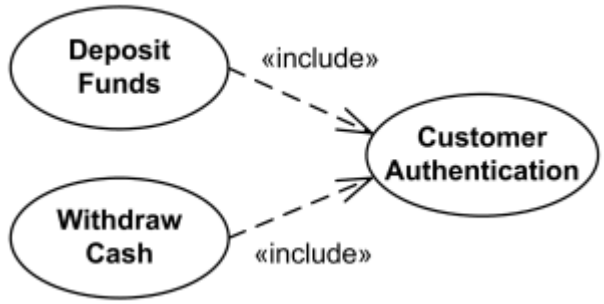


Figure 3.11 - Example of an <<include>> relationship in a Use Case Diagram

The Use Case Diagram for the current project encompasses the High-End alternative solution, designed for the project. There are four actors involved in the system: *User*, *Google Home IoT*, *E-book store* and *AI third-party software bot*. A couple of Use Cases are used by the multiple actors, like *Login*, *Browse external e-book database* and *recommend a new e-book*. The Use Case *Login* has the extension *Registration and profile creation,* which is only used for first-time users of the system. *Open e-book with voice command through mobile device* opens the book, chosen by the user, and has an option of *Place bookmark* and *Identify bookmark number*. The analysis of the possible alternatives will be discussed further in this chapter.

From the *Browse external e-book database* function, in which involved are the user and the e-book store, the user is required to go through many processes in the system: from *Select book to download* to *Open e-book with voice command through mobile device,* to *Turn to next page with voice command through mobile device,* while this is happening, the function *Track number of pages per hour turned* will start, then *Calculate number of hours needed to complete the volume of the book* and finally, from this *Recommend a new book* using the AI third-party

software bot, the process is then brought to *display new books recommendation* and back to the *Select book to download,* which begins the process again.
The application's Use Case Diagram is presented in *Appendix A3*.

3.4.2 Sequence Diagram

The Sequence Diagram is an interaction diagram that details how operations are carried out in a system. High-level interactions are presented between the system users (Actors) and the system objects. The purpose of a sequence diagram is to:

- model the interactions between object instances within a Use Case.
- show the interaction between objects within a collaboration that realises an operation.
- show elements as they interact over time.

Sequence diagrams are organised according to objects which are shown across the horizontal x-axis, and time which is presented along the vertical y-axis.
In a Sequence diagram, an *Actor* is the *role* played by external to the system entity, that interacts with entities inside the system. It is external to the system itself and represents roles played by human users or external systems. The *Actor* interacts with the *System Boundary* only, and not with the system objects themselves.

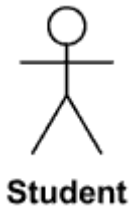


Figure 3.12 - Example of an Actor in a Sequence Diagram

The System Boundary is used to receive the actor's messages and re-sending them to the System Control, and for displaying the results of the functions taken from the system objects to the Actor.

Figure 3.13 - Example of a System Boundary in a Sequence Diagram

The System Control is responsible for managing the internal workings of the system. It determines what persistent entity receives what message and forwards messages to the object needed for processing it. It also returns messages from objects to the System Boundary to be displayed to the Actor class.

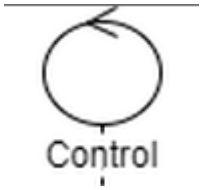


Figure 3.14 - Example of a System Control class in a Sequence Diagram

The entity classes are responsible for showing the persistent objects within the system. These objects are converted to object classes at a later stage of the Design phase. These entities receive messages from the System Control, which are from the Actors outside the system, and they then return the result to be displayed by the Boundary class.

Figure 3.15 - Example of an Entity class in a Sequence Diagram

Messages in a Sequence Diagram relate to the requests, made by objects, and are travelling through the system to be processed. They are marked on the lifelines of the objects and rewritten once they arrive to a specific entity. Messages can be forwarded to the same object they came from, which is known as a self-message. An U-turn arrow points to the destination the message will end up as it is being forwarded.

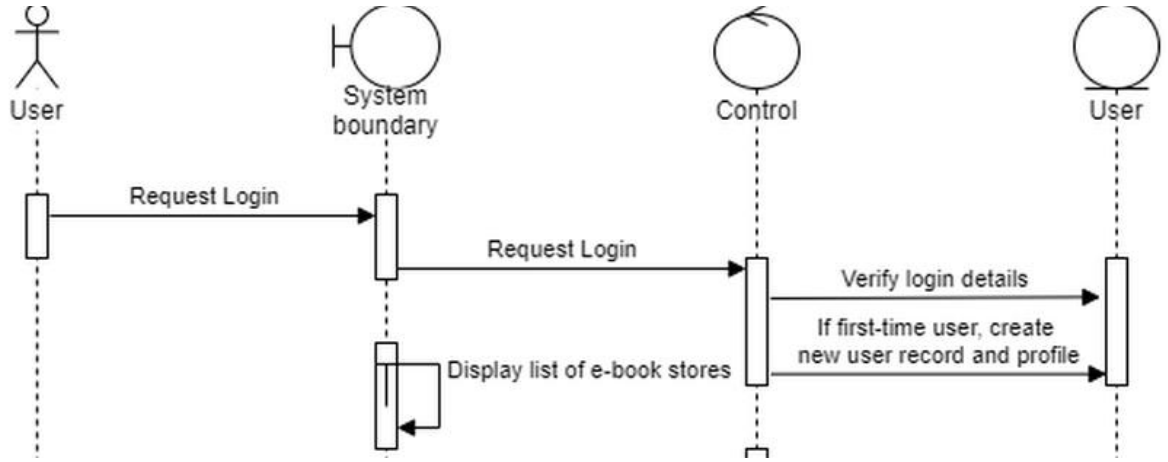


Figure 3.16 - Example of messages being sent from objects to entities in a Sequence Diagram

The Sequence Diagram for the current project is presented as follows:

Actors:

- User
    - Messages:
        - Request Login
        - Select e-book store database
        - Select e-book store genre
        - Connect to smart device Google voice assistant
        - Request download of specific book via voice command
        - Request opening the book via voice command
        - Request turning page of the book via voice command
        - Request creation of page bookmark to terminate reading session via voice command
        - Request start of new reading session by opening selected book via voice command
        - Request turning on the bookmarked page from previous reading session via voice command
        - Request closing of completed book
- AI third-party interface

Persistent Entities:

- User – holds user's information
- E-book store database – holds e-book information from online stores
- E-book store database genre – contains list of genres of e-books
- E-book store genre book – contains information on books in a specific genre
- Selected book record – contains information of the books downloaded by the user from an online store
- Page bookmark – contains bookmark page number of the e-book.

Representation of the Sequence Diagram for the current project can be found in *Appendix A4*.

3.4.3 Object Class Diagram

An Object Class Diagram shows the structure of the designed system at level of classes and interfaces, with their features, constraints, and relationships. The representation of a class is

shown as a rectangle with 3 sections: the top section contains the object's name; the attributes contain private information that must be held in the class (denoted by a – sign) and are presented in the mid-section, methods are lifted from the messages of the Sequence diagram and are listed at the bottom section with a + sign, preceding them.

The lines connecting classes with each other are known as *relationships*. The number placed near a class indicates the cardinality of the relationship, meaning there may be zero instances of a class to one instance of the other class, or one instance to many (Fakhroutdinov, 2021).

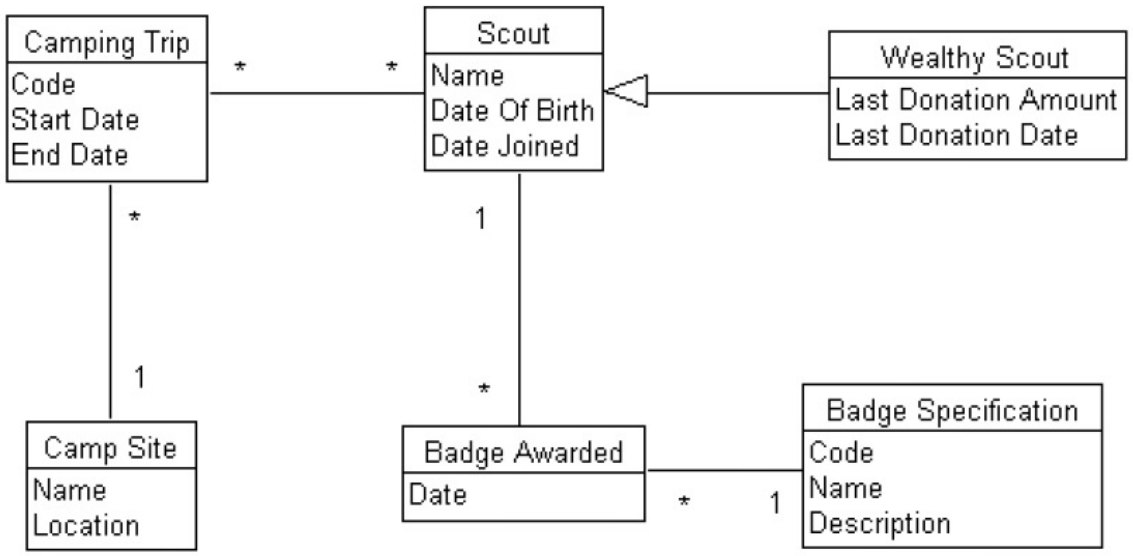


Figure 3.17 - Object Class Diagram (uml-diagrams.org, 2021)

The Object Class Diagram for the current project contains the following classes:

E-book store genre book

Attributes:

- BookID: Primary Key
- storeID: Foreign Key
- EbookStoreGenreId: Foreign Key
- bookTitle
- bookAuthorId: Foreign Key
- bookCoAuthorID: Foreign Key
- bookGenres

Methods:

- getSelectedEbookStoreDatabaseGenreBook

Author

Attributes:

- AuthorID: Primary Key
- authorName
- authorSurname

E-book store database

Attributes:

- storeID: Primary Key
- storeName
- storeURL

Methods:

- getSelectedEbookStoreDatabase

E-book store database genre

Attributes:

- EbookStoreGenreID: Primary Key
- storeID: Foreign Key
- bookGenreName

Methods:

- getSelectedEbookStoreDatabaseGenre

Selected Book Record

Attributes:

- selectedBookRecordID: Primary Key
- bookID: Foreign Key
- storeID: Foreign Key
- genreID: Foreign Key
- progressStatus

Methods:

- createNewRecordForSelectedBook
- ifLastBookPageReachedUpdateBookStatusComplete
- getRequestedBook

- ifLastBookPageReachedUpdatedBookRecordWithStatusCompleted

Page Bookmark

Attributes:

- bookmarkNumber: Primary Key
- userID: Foreign Key
- selectedbookRecordID: Foreign Key
- pageNumber

Methods:

- createNewRecordOfPageBookmarkForLastReadingSession
- getLastRecordCreatedForPageBookmark

User

Attributes:

- userID: Primary Key
- username (e-mail)
- password
- preferredGenres

Methods:

- verifyLoginDetails
- createNewUserProfile

Smart Device Google Assistant Interface

Methods:

- getConnectionWithGoogleCoiceAssistant
- gownloadSelectedBookViaVoiceCommand
- openSelectedBook
- turnPageOfSelectedBook
- createPageBookmark
- voiceCommandToOpenRequestedBook
- turnBookOnRequestedPageAsFromLastedBookmark

- turnPage
- voiceCommandToClose

The Object Class Diagram for the current project is presented in *Appendix A5*.

3.5 Alternative Design Solutions

Low-End alternative:

The Low-end alternative offers the very basic functionalities of the system which it cannot work without.

Features:

- Login
- Registration and profile creation
- Selection of book genre
- Browse external e-book database
- Select a book to download
- Open the e-book from the app
- Turn to next page or previous page with voice command through the mobile device

Mid-Range alternative:

The Mid-Range alternative offers a compromise between the Low-end and High-end alternatives. It includes all the functionalities of the Low-end solution, plus the following

Features:

- Place a bookmark at the end of a reading session
- Open the book and go to the *bookmarked page* with voice command through the mobile device

High-end alternative:

The High-end alternative includes the functionalities of Low- and Mid-range alternatives, plus some extra *Features* (that would complete the software project ideally):

- Track the number of pages turned per hour
- Calculate how long it will take to finish the book
- Recommend new books

3.6 Decisions Made for the Current Project

From the discussion at the beginning of the chapter, the *Android* OS easily surpasses the market share of the iOS in Ireland. *Android* is the most widely used operating system in the world for smartphones, which makes it more than important to develop for it. As the developer of the current project has more experience with *Java* development than with any language with iOS the focus will be on using the development environments that come with the *Android*, rather than the ones created by *Apple*. Using *Android* also benefits by the fact that its *ecosystem* is more open with voice assistants, than any other.

Since the current project has development time constraints posed by limited timespan of the project, the *Rapid Application Development* Methodology is chosen as most appropriate for the project through use of rapid and iterative prototype development.

The chosen Alternative Design solution for the current project is the *Mid-Range*, as it focuses on the most important aspects of the application, and due to the time constraints will be the most realistic one to be completed within the stipulated timeframe for the project.

3.7 Chapter Summary

In this chapter, discussed were different mobile operating systems, voice assistants, software development methodologies, and the Object-Oriented environment characteristics, with Unified Modelling Language diagrams and Alternative Design solutions for the project.

From the research conducted, it was established that the *Android* Operating System leads above the iOS market share in usage. It was also determined that to create a strong application it is necessary to involve Object-Oriented Programming into the coding process and to specify the best Development methodology to be followed. The Unified Modelling Language diagrams created for the project's Conceptual Model gave the developer a better understanding of the requirements for the project, as well as made it possible to design for different scenarios

(Alternative Design Solutions) that could be accomplished depending on the project's constraints.

The next chapter will describe the Design and Coding process of the current project's application. It will present the step-by-step process of building the application.

References

Ambler, S. (2002). *Agile Modeling: Effective Practises for eXtreme Profgramming and the unified Process.* New York: John Wiley & Sons. [Accessed January 19, 2021]

Android. (2021). *Codenames, Tags and Build Numbers.* Retrieved from Android.com: https://source.android.com/setup/start/build-numbers [Accessed January 19, 2021]

eternalsunshineoftheismind. (2013).*Traditional Methods of Software Development.* Retrieved from eternalsunshineoftheismind.wordpress.com: https://eternalsunshineoftheismind.wordpress.com/2013/02/08/traditional-methods-of-software-development-the-waterfall-model/ [Accessed January 19, 2021]

Fakhroutdinov, K. (2021). *UML Use Case Diagrams.* Retrieved from uml-diagrams.com: https://www.uml-diagrams.org/use-case-diagrams.html [Accessed January 19, 2021]

Geeksforgeeks. (2021). *Software Engineering Spiral Model.* Retrieved from geeksforgeeks.org: https://www.geeksforgeeks.org/software-engineering-spiral-model/ [Accessed January 19, 2021]

Gomaa, H. (2011). *Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures.* Cambridge: Cambridge University Press. [Accessed January 19, 2021]

gstatcounter (2020, December 29). *Mobile Operating System market Share Worldwide.* Retrieved from StatCounter.com: https://gs.statcounter.com/os-market-share/mobile/worldwide [Accessed January 19, 2021]

Guru99. (2021a). *Spiral Model: When to use? Advantages & Disadvantages.* Retrieved from Guru99.com: https://www.guru99.com/what-is-spiral-model-when-to-use-advantages-disadvantages.html [Accessed January 23, 2021]

Guru99. (2021b). *Agile Methodology: What is Agile Software Development Model?* Retrieved from Guru99.com: https://www.guru99.com/agile-scrum-extreme-testing.html [Accessed January 19, 2021]

Lucidchart. (2021). *4 Phases of Rapid Application Development Methodology.* Retrieved from Lucidchart.com: https://www.lucidchart.com/blog/rapid-application-development-methodology [Accessed January 19, 2021]

O'Carroll, B. (2020, March 20). *What is Rapid Application Development (RAD)?* Retrieved from Codebots.com: https://codebots.com/app-development/what-is-rapid-application-development-rad [Accessed January 19, 2021]

Product Plan. (2021). *Dynamic Systems Development.* Retrieved from Productplan.com: https://www.productplan.com/glossary/dynamic-systems-development-method/ [Accessed January 19, 2021]

Santos, J. M. D. (2021).*Agile Software Development Methodology & Principles.* Retrieved from Project-management.com: https://project-management.com/10-key-principles-of-agile-software-development/ [Accessed January 19, 2021]

Sinickl, A. (2019, August 10). *I want to develop Android apps - What languages should I learn?* Retrieved from Androidauthority.com: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/ [Accessed January 19, 2021]

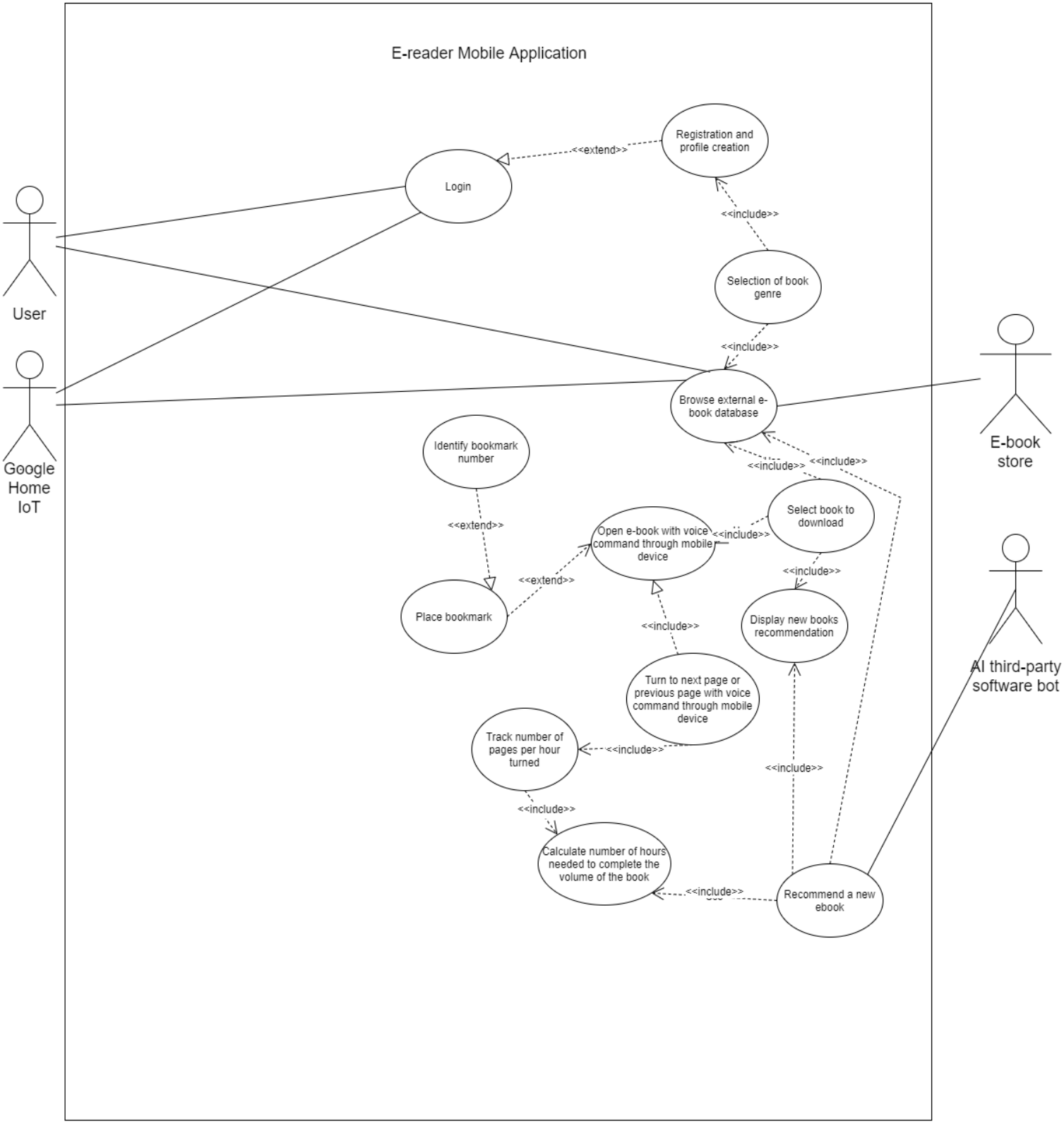
Somashekara, M., Guru, D., & Manjunatha, K. (2017). *Object Oriented Programming with Java.* Dehli: PHI Learning Ltd. [Accessed January 19, 2021]

T4 (2020, August 16). *Voice Assistant Market Share.* Retrieved from T4.ai: https://www.t4.ai/industry/voice-assistant-market-share [Accessed January 19, 2021]

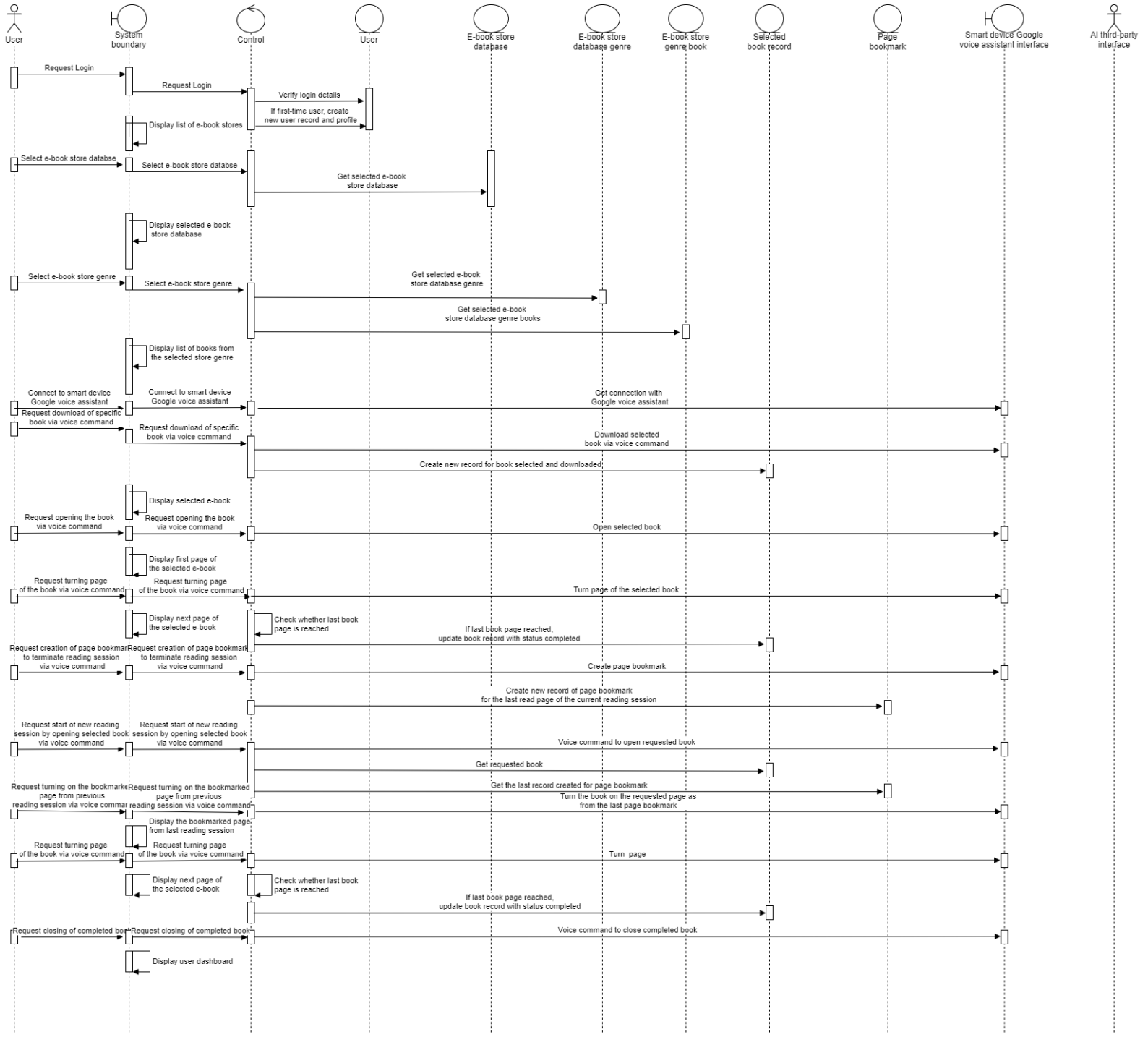
Uml-diagrams.org (2021, January 19). *Object Class Diagrams.* Retrieved from uml-diagrams.org: https://www.uml-diagrams.org/class-diagrams-overview.html [Accessed January 19, 2021]

Wikiwand. (2021, January 19). *Dynamic Software Development Model.* Retrieved from Wikiwand.com: https://www.wikiwand.com/en/Dynamic_systems_development_method [Accessed 19 January 2021]

Appendix A5: ***Project's Use Case Diagram***



Appendix A6: *Project's Sequence Diagram*



Appendix A7: *Project's Object-Class Diagram*