Software Design and Development of an Appointment Booking System: A Design Study

Kamarin Merritt[1] and Shichao Zhao[2][0000-0002-7048-8860]

[1] Northumbria University, Newcastle-Upon-Tyne NE1 8ST, UK
[2] Glasgow School of Art, Glasgow G3 6RN, UK
s.zhao@gsa.ac.uk

Abstract. This paper outlines, utilises and reflects upon effective software design and development through effectively applying current practical solutions such as use case modelling, class and sequence diagrams and by developing a prototype program which reflects the designs from said diagrams. A discussion is presented regarding the best method of approaching the Software Development Life Cycle (SDLC), justifying the selection and also concluding with a reflection on the professional, ethical and security issues in relation to the proposed design and use of software within the healthcare sector. In regards to ethical considerations, the Code of Ethics was discussed in relation to the design and development of the system and the importance of it alongside ensuring a high level of ethics and professionalism is of utmost priority when handling personal identifiable information in the development of a system. As well as this, security issues during and post development is an area of significant importance when it comes to data management, this has been discussed in general and against the software development life cycle. Additionally, in the design and development of software it is crucial to abide by world software quality standards such as International Organisation for Standardisation (ISO), and this paper reflects upon current issues and how to address these in the process of design and development.

Keywords: Systematic Design; Digital Information Management System; Design Reflection.

1 Introduction

The purpose of this paper is to design and develop a program prototype solution for a case study, which will see the development of a hospital appointment booking system and develop a critical discussion on the current issues. This paper will demonstrate a Use Case Diagram which will model the functional requirements from the case study and document one primary use case. Secondly, a Class Diagram will be produced to model the systems structure, displaying various classes and their variables/attributes, relationships and procedures. Thirdly, a Sequence Diagram will be produced, illustrating an overview of the functional requirements for one of the primary use cases, and this will be concluded with a prototype program, showing a proposed solution using OOP language (object-oriented programming).

Furthermore, this paper will go in depth and critically discuss the Software Development Life Cycle (SDLC) methodology adopted in the solution development and critically reflect upon the professional, legal, social, security and ethical issues related to the design and development of the solution developed for the case study.

This case study is based around a hospital, who has built a new central dental unit and they now want to create a digital information management system to replace the outdated one. This will fulfil the purpose of recording, searching, updating and storing patient data. It is expected that all staff will use this system in order to support day-to-day tasks such as booking, deleting, updating, managing and recording appointments, while also allowing for the system to be utilised for other business needs. The system needs to ensure efficiency and ease of use, through two functions (booking online or via call). Additionally, the system will need to accommodate many functions.

2 Related Work

Typically, a use case is used to capture the requirements of a system and what the system is supposed to do [1]. When building this system, the first step we take in the Object-Oriented Modelling (OOM) approach is to build a Use Case Diagram. This stage is crucial as starting with a 'use case subject' allows for key functional requirements, such as logging in and out, to be defined and outlined [2]. One of the many benefits of using models to document functional requirements, is that it enables you, as the system analyst, to define clearly the details of the needs and requirements of the stakeholders, thus allowing for ease when explaining to stakeholders a system solution or proposal [3, 4]. The overall purpose of a use case diagram is to identify the Functional Requirements. Requirements identify a list of characteristics that a system must have and also describe what a system should do; in software engineering, there are two types of requirements, which are; Functional Requirements and Non-Functional Requirements [4]. Functional requirements identify potential 'features' in which the given system may have [4]. Therefore, a list has been generated, of the functional requirements which derive from, the case study: Login; Logout; Handle Login Errors; Check Account Balance; Book Appointment; Cancel Appointment; Check in Patient; Review Treatment History; Amend Appointment; Available Tools and Equipment; Transport History. The above Functional Requirements extend further in the Use Case and are discussed in the further section.

In all use cases there is a person who will use the system, this person is known as an 'actor' [3]. This actor specifies a role which is performed by the user. An actor models an entity which is outside of the system, and it is crucial to identify who these actors are as they play a role in interacting with the system, to make it work [3]. In the case study, there are various actors who fall under different categories, due to their relationship with the system. To define these actors, it is suggested that we need to ask ourselves as a systems analyst, the following: Who will supply, use or remove any information? Who will use the function (i.e., who will login, logout, etc.)? Who will support and maintain the system? What do the systems need to interact with? What does the actor need to do exactly? [2]. By following this criteria, the actors within the

specific case study, and those who will interact with the use case, are: User; Patient; Staff; Doctor; Receptionist; Admin; Practice Manager; Driver. It was stated that there are two types of actors: Primary Actors and Supporting Actors [5]. All of the actors within this overall use case subject, are Primary Actors, as they are all a stakeholder who calls upon the system, to deliver one of its services. In the section below, actors are discussed in more detail regarding relationships with other actors and classes.

3 Design Process

This section will go in more depth when it comes the use case and relationships each actor and the use cases have, while also applying and documenting them via diagrams. The overall purpose of the design process is to show how each element of the process is mapped out in accordance with the earlier use case requirements. Relationships, Inheritance, Use Cases and Class Diagrams were developed in StarUML and this was then followed by a prototype solution being written in code and developed in NetBeans. StarUML is a Unified Modelling Language (UML) software tool which allows for agile and short prototyping, which is usually targeted to be used by small, yet agile teams. Additionally, NetBeans is the tool which allows for the StarUML prototypes to come to life through, as it is an open source, integrated development environment that is meant for the development of applications on various operating systems. Below, each stage has been discussed in depth and processes have been applied in depth also.

3.1 Relationships in the Use Case (Design Decisions)

There are various types of relationships when developing a use case, and these can be between actors and the use cases themselves [4]. Below, is a breakdown of the relationships which have been utilised in the Use Case shown in Figure A1.

Actor Inheritance. It is highly common for all systems to have a generic User, in which other actors inherit attributes from [4]. The purpose of having a generic user is so that within diagrams such as the Class Diagram, classes can be illustrated simply, rather than doubling up information in a class, we can use a generalisation relationship; also known as a parent-child relationship [1]. In Figure 1, there is an outline of the relationship between each actor, the clear headed arrow indicates a generalisation relationship.

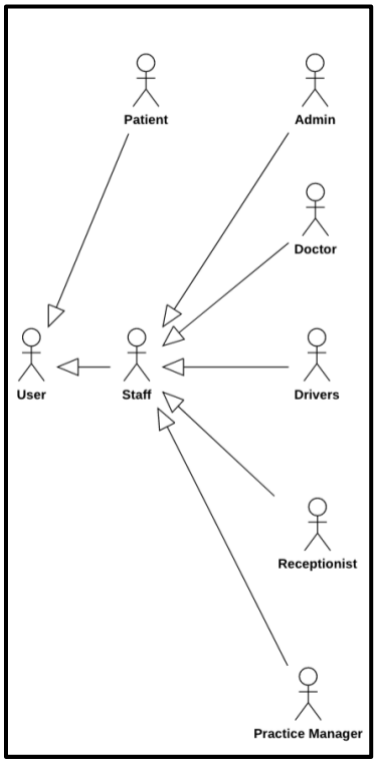


Fig. 1. Actor Relationships in the case study.

Use Case Relationships. Within the Use Case diagram, various use cases have a relationship between them, which allows for the functionality of the system to be demonstrated clearly, for stakeholders to grasp with ease [5]. The most common relationship between use cases is the directed relationship, more specifically 'extend' and 'include' as shown in Figure 2. In Figure 2, we can see the use case 'Book Appointment', for example. Now in the new system being developed, patients will be able to book an appointment through two different methods, either via calling in; or booking online, the purpose of <<extend>> is to allow users within the system, a choice. However, if the patient is not registered, their path in the process would be different than that of an existing patient. Therefore, in order to enable this option, we would have 'Book Appointment' as a main use case and then we use the directed relationship (<<extend>>) to get to the 'Register Patient' stage in the system, which would allow us to add the patient to our system after taking relevant details, then they can follow the path of an ordinary patient by choosing their preferred booking method. As we can also see in Figure 2, there is an <<include>> relationship, now this is a required function that needs to be undertaken, unlike the <<extend>> relationship and a user in the system cannot proceed to the next stage or use case, without undertaking this function. This has been explained more clearly, in the section below which demonstrates the 'Book Appointment' use case in more detail.

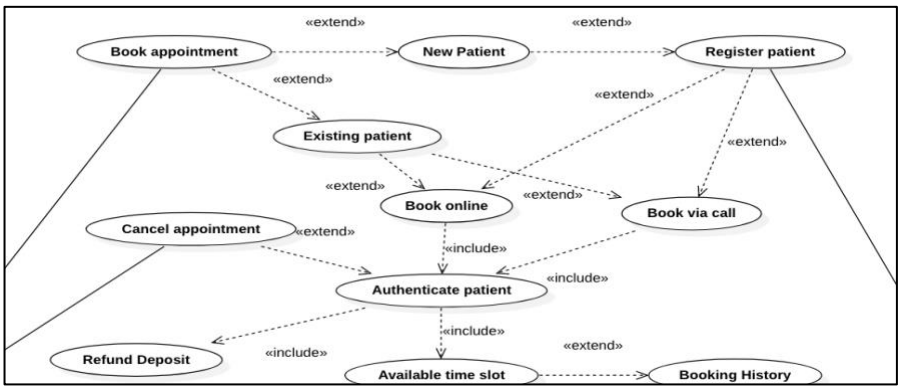


Fig. 2. Screenshot from the system Use Case Diagram.

Documentation of a Use Case using House Style. In order to demonstrate various elements of a use case within the Use Case subject, the 'Book Appointment' use case has been documented through using the House Style technique (as shown in Table 1). The purpose of documenting this particular use case is due to the fact that it is one of the key use cases within the system, also the purpose of this is to illustrate further details that may not be visible in the use case diagram.

Table 1. Documentation and structure of the 'Book Appointment' use case diagram.

Name: Book Appointment
Description: Patient wants to book an appointment, which can be done via calling in or booking online. Admin will determine date and time, upon which a deposit will be taken, and confirmation notification will be sent to the patient.
Actors: • Patient • Administrator • Receptionist (in the case of Cancelling appointment or no show)
Pre-Conditions: • Be a registered patient • Patient credentials have been validated (booking via call and online) • Patient needs to book an appointment

Basic Flows:
• Use case begins when Actor contacts to book appointment
  o << extend >> Book via call
  o << extend >> Book online
• Authenticate patient
• Determine appointment date and time
• Deposit payment
• Appointment confirmation notification

Post Conditions: • Patient successfully books appointment slot • Administrator saves record of appointment and updates patient record • Patient receives appointment confirmation notification
Alternative Flows: • If patient is not registered, admin team will have to register patient before booking appointment • If patient authentication is failed, terminate call (via call) or display error message and return user to home screen (via online) • Call lines full – patient will need to hold the line and wait (via call)
Notes: --
Business Rules • Patient must be registered before booking an appointment • Patient must verify details • Patient must have required amount of credit within their account to pay deposit (usually a notification is sent to the patient if balance is low

3.2 Class Diagram (use STARUML)

The class diagram which illustrates the system is extremely complex and has many classes within. Therefore, no design pattern has been adopted, as it would cause confusion within the system. Using design patterns in UML may cause for confusion or lack of function [4, 6]. In the class diagram, there are various main classes, the most important one being the 'Appointment' class. The reason for this being the main class is that it is connected to most actors and requires an interaction from various aspects of the system such as payment, notifications, and actors such as doctors and admin. This class has various associations with other classes, as shown in the snippet in Figure 3 on the left side and full diagram in Appendix 2, the main type of relationship is composition. The composition relationship is similar to a parent and child [2]. The child cannot exist without the parent, just like a room cannot exist without a house [2, 4]. Therefore, by connecting other classes with the composition tool, this shows that for example, a 'Confirmation Notification' cannot exist without booking an appointment.

Furthermore, multiplicity notations have also been utilised, the purpose of this is to indicate the number of instances of one class linked to one instance of another class [4]. For example, as seen in Figure 3 on the right side, one appointment will have one payment, but one payment can only be made from booking one appointment, thus the reason for the '1' notation at either end of the composition relationship. It has been highlighted that in a class diagram, 'multiplicity' is when the association between classes is being depicted [4]. For example between 'Account' and 'Patient', there can be one to many (1..*) patients, but only one (1) account per patient. Therefore, along the association relationship line as shown in Figure 3 on the right side, we can see this illustrated. The diagram is not limited to the above specific relationships and multiplicities, and many others have been used and can be seen in the full diagram outlined in Appendix 2.

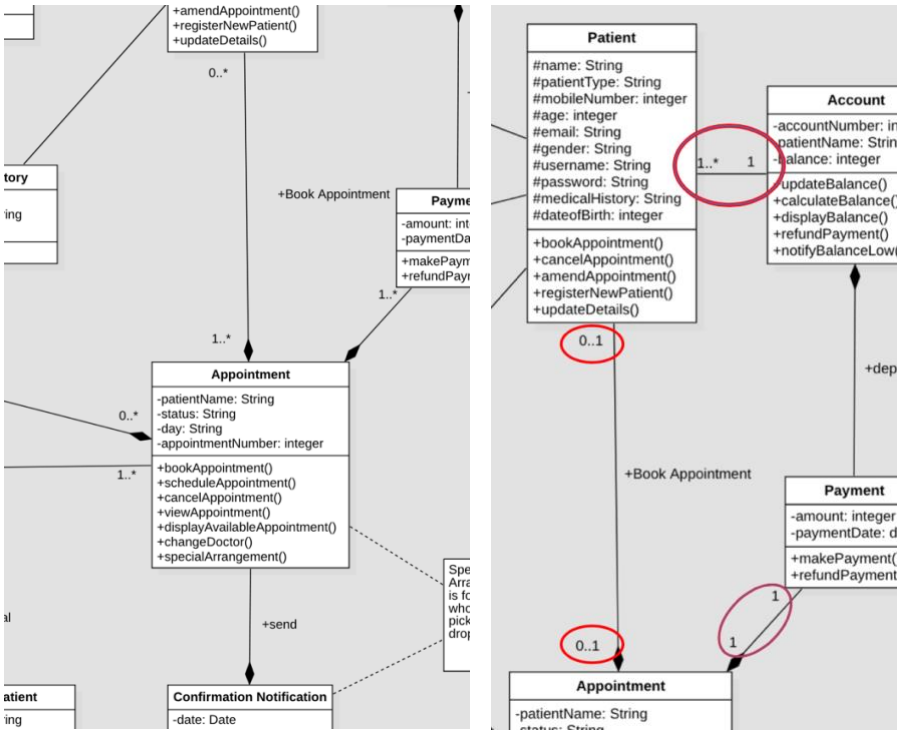


Fig. 3. Screenshot from the Class Diagram, full diagram can be found in Appendix 2.

3.3 Sequence Diagram (use STARUML)

In order to further demonstrate the Use Case documented earlier using the House Style documentation method, a sequence diagram demonstrating this Use Case 'Book Appointment' and the basic flows of it, has been developed in StarUML and is illustrated in Appendix 3. A Sequence Diagram is a UML diagram which demonstrates how the system or the classes within a code interact with one another [1]. The new system requires a core function which is to book appointments for patients, and the sequence diagram demonstrates this through mapping out a sequence of interactions to perform this function.

This sequence diagram demonstrates the steps of the system when booking an appointment over the phone and online, however, the diagram firstly outlines a Login sequence which is paramount for when the user, in this case the Admin and the Patient, want to access the online system and when the Admin wants to use the system to record an appointment (see Figure 4). Thus, the two main sequences are Login and Book Appointment, as booking an appointment could not happen without the Admin logging into the system. The main sequence in this diagram, however, is the appointment booking sequence which is shown in Figure 5. In this sequence there are lifelines which are vertical dashed lines, in which various objects (i.e., Booking History; Patient

Records; Account) and Actors (i.e. Patient; Admin) connected by various different interactions, are placed. This helps show the process overtime, moving down the lifeline means that more time is passing [4]. A 'Message' connects lifelines together and represents an interaction, for example veryDetails (dob, name).

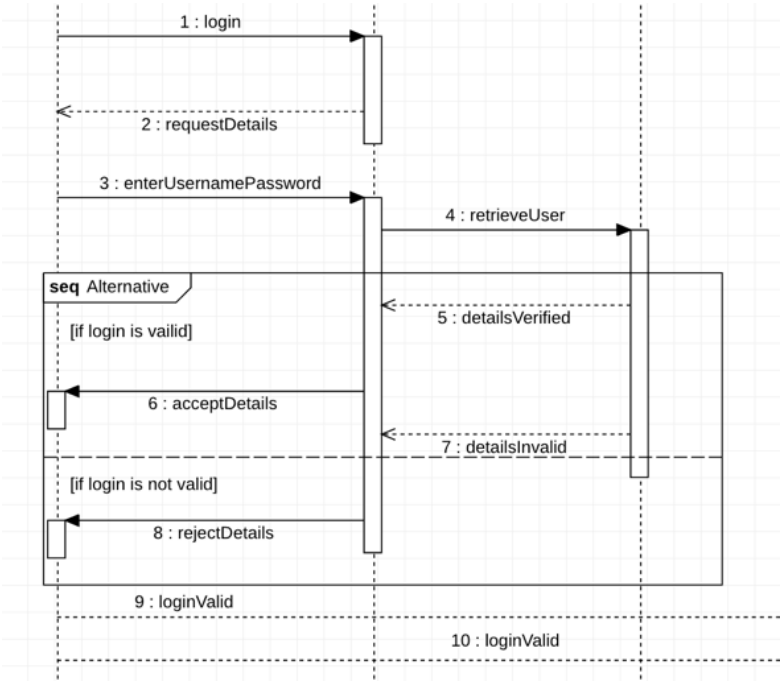


Fig. 4. The login sequence extracted from Sequence Diagram; full diagram shown in Appendix 3.

In the Appointment Booking sequence diagram, the first step (after successfully logging in) is to book an appointment. In Figure 5 we can see the first interaction is 'bookAppointment' illustrated by a straight line with a filled in arrowhead, this interaction is known as a message and is sent from one actor to another to start the process. Below this is a further interaction, known as a reply and is represented by a dashed line. The first sequence is broken down into two stages, as it is only possible to book an appointment if the patient is already registered, if they are not registered, they will not be able to book an appointment and follow the normal sequence of the diagram which includes steps interactions 11-16. Therefore, an 'alternative' frame is used. This symbolises a choice between two message sequences, which are usually mutually exclusive, and this has been placed around interactions 11-19. The conditions are if the patient is registered or if they are not. A dashed line separates the two choices, the option above is called 'patientExists' and will proceed to the following sequence of booking appointment, however the below option is called 'patientNonExistent' and will extend the sequence to an alternative path to register them. The two main interactions used in the sequence diagram are message and reply message, with the design decision

to use various alternative frames to navigate through different areas of the system, when a function cannot be performed.

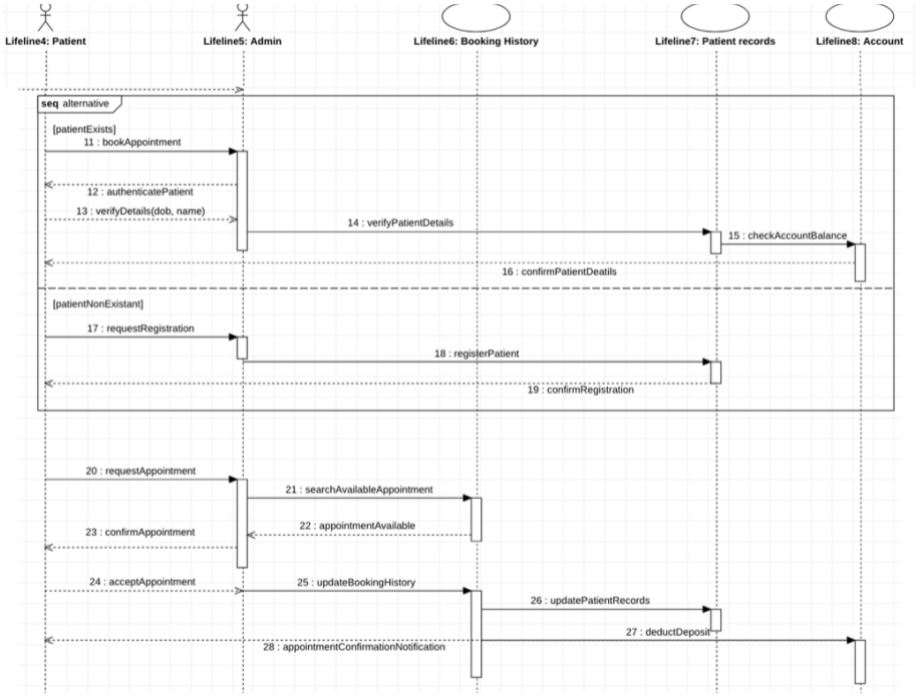


Fig. 5. The login sequence extracted from Sequence Diagram, full diagram shown in Appendix 3.

3.4 Program Prototype

As this prototype will become extremely long due to the nature of the system, there has been a focus on the main class of 'Login' and 'Book Appointment'. The system prototype reflects how a patient would log into the system and book an appointment via the online method, outlined originally in the Use Case Diagram and Sequence Diagram. The first JFrame (a JFrame is a container which provides a window, in which other components such as menu, buttons, text fields etc rely on) is built around the 'Login' class. In the class diagram, the User was a generic actor and would hold the operation of being able to login to the system, upon verification of their details. In the prototype developed in NetBeans, upon inserting Graphical User Interface (GUI) components (i.e., Buttons, Text field etc), raw code and strings were connected to them (see Figure 6) to allow for specific credentials to be used in order for access, and if not, the user would be presented with a 'Login Failed'. This was possible due to using the code 'String name/pwd =' and setting and 'if' code to verify what name and password equal. If the login is successful, the user will be presented with a 'Login Successful' alert on the login page and an additional JOptionPane will pop up, allowing the user to progress to the next screen, this can be seen in the code too (see Figure 6 & 7). In order to connect

the login page JFrame to the next page, known as ‘patientAccount’, a connection had to be made. This was implemented by setting up the GUI button to register an event, creating a ‘java.awt.event.MouseEvent evt’ private void. In this event we had to add code to make the connection work, and this can be seen within the code. To make the transition more fluid, the use of ‘this.dispose’ was used, so that the previous window would close, and the user wasn’t left with multiple open windows – this helps fluidity when it comes to progressing through an application so multiple windows are not left open. The use of ‘this.dispose’ is beneficial as it free’s up allocated memory or releases unmanaged resources.

```
    private void loginButtonActionPerformed(java.awt.event.ActionEvent evt) {

//login credentials, including else for failures

    String name = usernameTextField.getText();
    String pwd = new String(passwordField.getPassword());
        if(name.equals("Kamarin") && pwd.equals("User1"))

{
    JOptionPane.showMessageDialog(null, "Login successful, welcome " + name, "Successful Login", JOptionPane.INFORMATION_MESSAGE);
        new patientAccount().setVisible(true);
        this.dispose();
}
    else
        JOptionPane.showMessageDialog(null, "Login Failed" + name, "Failed Login", JOptionPane.INFORMATION_MESSAGE);
    }
```

Fig. 6. Login Code, containing ELSE variants for login failure.

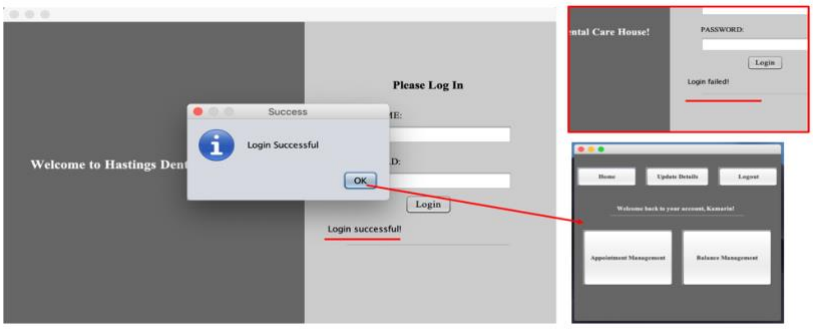


Fig. 7. Login JFrame in demonstration showing two results (Login.java).

Once at the next window known as ‘patientAccount’, the user is presented with options, all of which were set out the attributes/functions, both in the use case and class diagram. This JFrame included a link to the main class, ‘Book Appointment’ and to progress to this stage the user must simply click the Button GUI ‘Appointment Management’. Again, similarly to above and the Login button, code was used to allow for the current JFrame to transition to the next (see Figure 8).

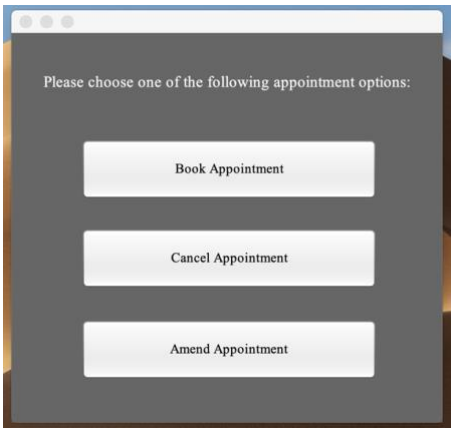


Fig. 8. Appointment Management JFrame, developed in NetBeans.

In this JFrame, the user is presented with three options. Due to the complexity of building three different JFrames and code, only 'Book Appointment' button will take you to a working JFrame, the other two options take you to a new, but blank JFrame. This is simply just a navigation page, and uses minimal coding to progress, the coding for this was already demonstrated earlier. Upon clicking the 'Book Appointment' button the user is provided with the final window of the system, which is 'bookAppointment' and they are presented with appointment slots which were connected in one Panel using various Check Box swing controls and then inserted into the swing control, List. Once the user has selected an appointment time, they are then able to press 'Book appointment'. After clicking this button, a JOptionPane will pop up confirming the booking (see Figure 9) and by clicking OK, this will end the process from the patient's side, and Admin will continue to finish the process of updating patient records/assigning doctor.

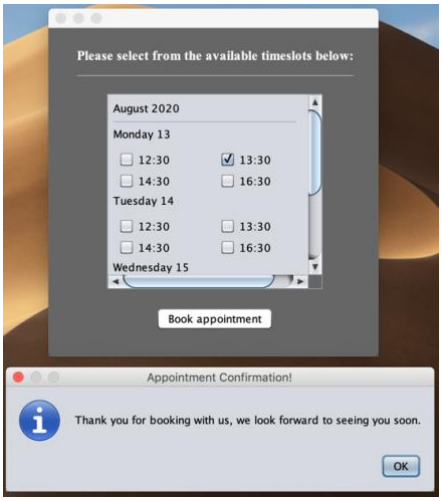


Fig. 9. Book appointment JFrame, also showing JOptionPane, developed in NetBeans.

4 Discussion

This section is going to critically discuss the Software Development Life Cycle (SDLC) methodology adopted in the above solution development and critically reflect upon the professional, legal, social, security and ethical issues related to the design and development of the above solution.

4.1 Critical Discussions: Software Development Life Cycle (SDLC)

The Software Development Life Cycle is a phased methodology which produces software to a high standard with low cost and the shortest possible time frame [1]. Depending on the chosen model, the SDLC provides a structured order of phases which enable an organisation to efficiently produce robust software that is both well tested and ready for use [4]. These phases for example, could be: 1. Planning; 2. Analysis; 3. Design; 4. Implementation; 5. Testing; and 6. Deployment and Testing [7].

There are a plethora of SDLC models, the most popular models are Waterfall Methodology and the Agile Methodology [1]. These are used massively, in order to produce an efficient and effective robust program solution. Thus, it was paramount that the correct SDLC was adopted prior to starting the process [4]. If the correct methodology is not used, this can cause the solution to become problematic within the process and cause design flaws, lack of security and disrupt the functionality [8].

The Agile methodology consists of 6 stages, as shown in Figure 10 and is an iterative and incremental based development, as it allows for requirements to be changeable according to needs [9]. Generally, the common issue in systems development is that there are constant requirement changes or even changes in the technology itself, however, Agile is a software development methodology which was intended to overcome this particular issue, therefore not managed correctly, the project may lose track and the process may be disrupted [10].

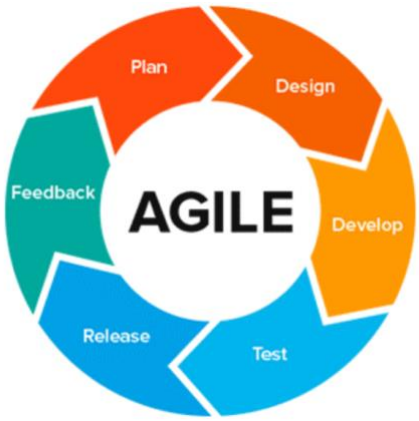


Fig. 10. Agile Methodology phases, adapted from Kendall and Kendall [1].

This presenting a limitation of using the Agile method, is that the project may lose track if not managed properly and the functions are not outlined correctly [11], whereas Waterfall methodology is an easier method to manage. Due to its nature, each phase has specific deliverables and a review process [4]. With Waterfall (as shown in Figure 12), it can be highlighted that its approach values planning ahead, keeping direction straight and accurate, however, if the requirement is not clear at the beginning, the method can become less effective, which will have a major impact on cost and implementation [12]. This presenting the major different between Agile, the fact that Waterfall approach values planning ahead but can meet more obstacles throughout the process, while Agile values adaptability and involvement and is able to change during the process [4, 11].

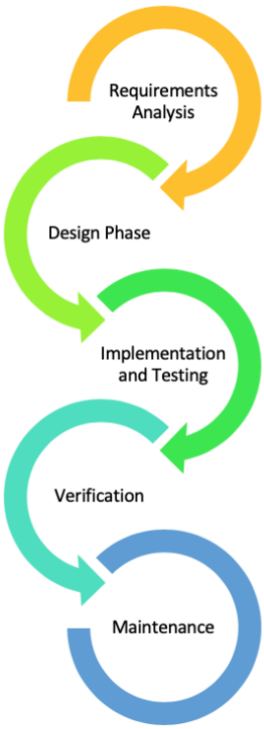


Fig. 12. Waterfall Methodology, adapted from Sommerville [4].

Furthermore, Waterfall has less stakeholder involvement which causes a hands-off approach. This approach is not suitable for every product, and the majority of users/stakeholders need to be involved in the system for example, to meet the correct requirements [12]. However, Agile focuses on delivering a high quality and user-friendly product through incorporating stakeholder engagement into all stages which allows for a more structured list of requirements [1]. Thus, Agile is more suitable in the

development of this system as the methodology has two core elements: Teamwork and Time – and breaks the overall project into smaller pieces with an individual time scale and ‘sprints’ [13]. Using such a method allows for feedback from previous phases to be planned into the next, so if a requirement is missing, or feedback has been given, this can be used, whereas in Waterfall, what is developed is based on the feedback and plans given in the first place [1].

4.2 Agile Methodology within the Program Solution

As discussed above, Agile is one of the most robust and beneficial methodologies to use in during the SDLC. The main benefit for using this methodology in the development of the program solution is that feedback can be given per stage and requirements can change along with way. The key instrument for Agile development is iterations, and this process sets out actions which are repeated until a condition is met [4]. One of the most productive Agile development tools is Scrum. Scrum uses ‘Sprints’ which allow for the development team to achieve and complete certain parts of the systems functionality. Therefore, as the program solution was only developed up to the stage allowing for the patient to book an appointment, this allowed for this portion of the solution to be robust (as shown in Table 2).

Table 2. Documentation and structure of the ‘Book Appointment’ use case diagram.

	Agile Software Development Life Cycle Phases
1. Plan	When developing the system, a use case diagram was developed, deriving from the requirements outlined in the case study. This allowed for the outlining of functional requirements which would be required in the system. After this, a Class Diagram was outlined showing various relationships between classes and also outlining various attributes and functions specifically, for each class. Thus, a Sequence Diagram was developed, and the Design Phase could take place. It has been noted by many that if developers decide to ignore this stage, the system will be prone to ‘feature creep’. This is where non crucial features keep arising and having to be implemented along the way, which while scrum is a good method to implement these along the line, it can cause for the projects more important tasks to lack time [1,13]

2.	Design	There is usually two approaches in designing the solution: 1) looking at the visual design i.e. JFrames, and; 2) looking at the architectural structure of the solution, i.e. the source code. In the design phase the coding was implemented in the Phase 3, but here, the basic JFrames were developed in NetBeans, as outlined earlier in Figure 7 and 8. In these JFrames, various GUI components were inserted per guidance of the previously developed diagrams. From making these GUI components in regards to the requirements, the system started to take place, however could not take place without building in code.
3.	Develop (Coding stage)	Now all GUI components were built into the JFrames, the functionality needed to be coded into the source code, in NetBeans.
4.	Testing	Unlike in the traditional SDLC approach where development and testing are individual phases, Agile connects these two phases together, allows for the testing and development team to work together as one and reduces overall costs and defects which may incur later [1]. NetBeans allows a program to be run, thus the ability to test if coding works and the GUI components work when used and the JFrames connect to each other, as supposed to. Agile Testing stage is to draw out any defects and fix them as soon as possible prior to releasing [4]. In the developing phase, coding appeared difficult when trying to build the various different JFrames for different users, therefore, only the Patient (Actor) portion was built. This is usually the longest process as it is writing advanced functionality of the system and allowing it to work. However, in the developing phase, there usually isn't many changes to the iterations here, this is in Phase 4: Testing. During Phase 4, further iterations become apparent as the factors such as integration, operability and user testing take place.
5.	Releasing	As in the above phases, only the patient portion of the booking system was built. This allows for the system to be released to users initially, in order to gain any potential feedback on whether the user interface is unappealing, hard to navigate and whether there are any additional requirements that could be built into the system when the sprint for the next round starts again. From this, iterations are updated in the existing software, which introduces additional features and resolves any issues [4].

6.	Feedback	Upon completing all of the previous development phases, the whole system can be reviewed. When finishing the solution in NetBeans, it was apparent that the coding and the JFrames held some functions that were not outlined in any of the Use Case, Class or Sequence Diagrams. Therefore, these were able to be updated accurately and made sense when looking at the diagrams and prototype in comparison to each other. After this final phase, the Agile software development life cycle could begin again in order to extend the current booking system to other actors such as Admin, Doctors and so on.

4.3 A Critical Reflection: Professional, Security and Ethical Issues

When it comes to the design and development of a system which involves a user interface and requires interaction, especially of that in the healthcare sector, there are a plethora of professional, security and ethical issues which may arise [4]. Within the development of the system, it was paramount that these issues were considered.

Ethical Issues. The term ethics refers simply to the governing, conduct and principles of people's behaviours [4]. In software development, ethical issues are massive and can be very complicated, this is due to the rise of new technologies such as machine learning, AI and Big Data. Some of these issues include privacy, transparency, accountability and diversity. However, in order to overcome these issues, it is imperative that organisations who develop and design their own software, have a Code of Ethics. Specifically, personally identifiable data in healthcare has been an arising issue in the past few years and following the Code of Ethics will allow for employees to understand and be trained on a set of acceptable behaviours in order to remain ethical and professional, when handling personal identifiable information, and organisational data in general [14], in software development, ethics is taking into consideration all of the implications which we may create as a result of software algorithms. As an organisation, it is imperative that standards are not violated and that as a software developer, systems analyst or even just an employee ethics and also morals are used interchangeably [14].

Security Issues. In the software development process, security is a constant process that involves all employees and practices within an organisation [15]. As technology advances, software environments are becoming more dynamic and complex, causing the development of systems to become more challenging. Also, with the increasing growth of the software development life cycle, the engineering behind it is under huge pressure to deliver requirements fast [15]. This causes less attention to be focused on security issues in which may arise in the system or software.

Many software applications are outsourced, and the development lacks in incorporating software security and due to this, issues arise around the growth of an

organisation and availability of meeting requirements for its customers [16]. Overall, from the be-ginning, organisations should ensure that during the software development life cycle, security is tightly bound from the get-go. The TCSEC Glossary (Trusted Computer Systems Evaluation Criteria) defines security requirements as a set of laws and practices which govern how an organisation protects, distributes and manages any form of sensitive information [17]. It also states that security requirements should be measured during the SDLC, and can be classified into functional security requirements, non-functional security requirements and derived security requirements [18]. Thus, during the SDLC, software developers must review these specific requirements outlined in the glossary, in order to reduce vulnerabilities down the line once the system is deployed.

Professional Issues. In the design and development of software, it is crucial to abide by the world software quality standards, in which there are various professional issues that may arise, the majority of these relate to the factors which are outlined in the ISO9126, such as: Functionality; Reliability; Usability; Efficiency; Maintainability; Portability [19]. When developing software in an organisational context, it is crucial to take into consideration the user and their experience. Firstly, user experience (i.e., Functionality; Usability; Efficiency) is crucial in gaining feedback on the systems from a user perspective [20, 21]. For example, this allows developers to reflect upon system feedback and incorporate new iterations in the program to make it more usable, functional and efficient for the user – thus, professional issues generally arise in the Testing stage of the software development life cycle, particularly in the Agile methodology [1]. Furthermore, when taking into consideration a patient booking appointment system, the system should cater for all users. Thus, factors such as disabilities, visual or hearing impairments and also those who may be illiterate are a key focus when developing and designing a system; allowing for everyone to book an appointment successfully. This would also include developing the system to not solely be based on JAVA and developing it for various platforms which would allow it to reach as many users as possible.

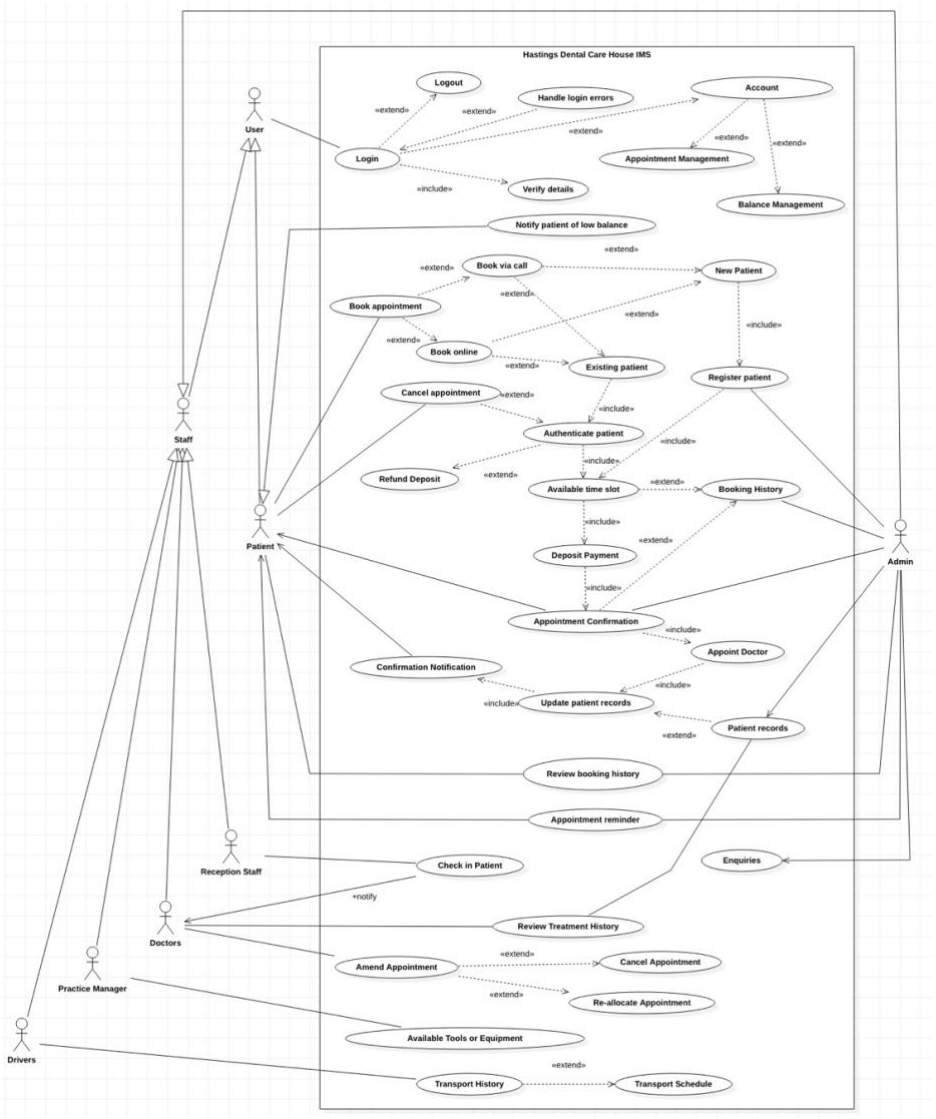
5 Conclusion

To conclude, this paper has developed from start to finish a program solution prototype for the case study. This has included outlining the functional requirements of the system, down to coding and creating the system itself in NetBeans. Upon doing this, the SDLC was reflected upon and how, as a systems analyst we were able to approach it. Furthermore, it was apparent that due to the vast amount of Ethical, Security and Professional issues in the software engineering domain and generally in organisations, that there are many principles and legislations that need to be followed in order to overcome these.

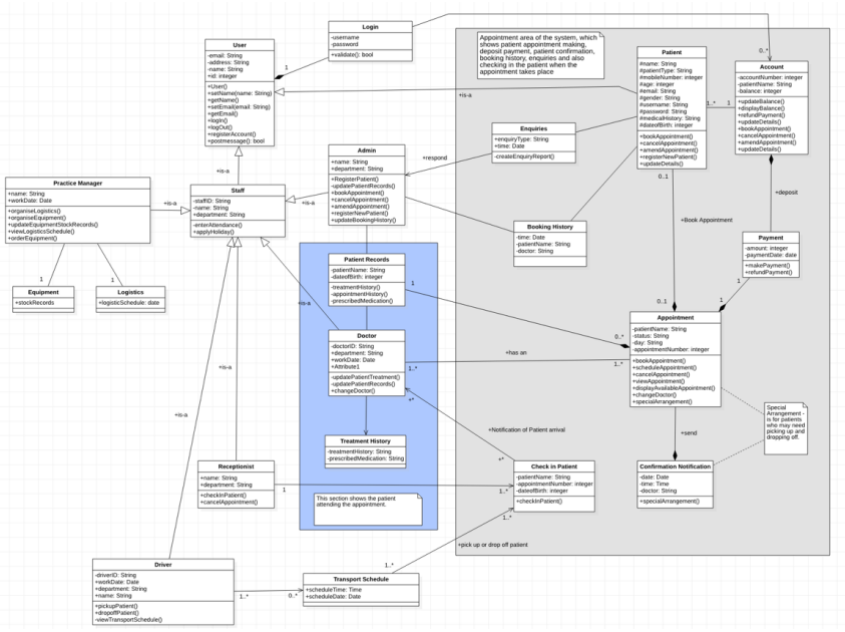
This paper enables useful insight into developing an appointment booking system through looking into the core, and various aspects within the SDLC, choosing the correct method of approach whether this is through the waterfall or agile methodology

and in general discussing the best method of approaching the SDLC, justifying it and reflecting upon various issues that may arise. This is an important element of contribution from this paper, as it approaches software design and development through effectively applying it, while also justifying and reflecting upon the effectiveness of using it within practical solutions. Additionally, this paper reflects upon current issues (i.e., ethical, security, legal, etc.) and how these can be addressed in the process of design and in general, development.

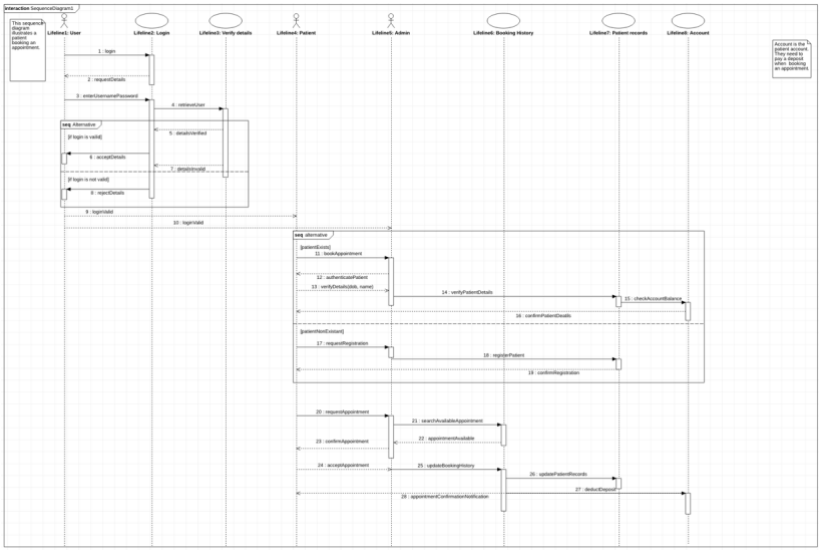
Appendix A



Appendix B



Appendix C



References

1. Kendall, K., Kendall, J.: Systems analysis and design. 10th edn. Pearson Education Limited, Harlow, United Kingdom, pp. 1–576 (2019).
2. Bruegge, B., Dutoit, A.: Object-oriented software engineering using UML, patterns, and Java. 1st edn. Harlow: Pearson Education Limited, Essex, United Kingdom, pp.11–40 (2014).
3. Satzinger, J., Jackson, R., Burd, S.: Systems analysis and design in a changing world. 1st edn. Cengage Learning EMEA, Andover, United Kingdom, pp.171–200 (2008).
4. Sommerville, I.: Software engineering, global edition. 10th edn. NOIDA: Pearson Education Limited, Uttar Pradesh, India, pp.119–140 (2016).
5. Cockburn, A.: Writing effective use cases. 1st edn. Writing Effective Use Cases, Addison-Wesley, London, pp. 1–259 (2006).
6. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. 1st edn. Addison-Wesley, Massachusetts, United States (1994).
7. Sharma, P., Singh, D.: Comparative Study of various SDLC models on different parameters. International Journal of Engineering Research. Comparative Study of Various SDLC Models on Different 4(4), 188–191 (2015).
8. Aljawarneh, S., Alawneh, A., Jaradat, R.: Cloud security engineering: early stages of SDLC. Future Generation Computer Systems 74, 385–392 (2017).
9. Andry, J.: Purchase order information system using feature driven development methodology. International Journal of Advanced Trends in Computer Science and Engineering 9(2), 1107–1112 (2020).
10. Rauf, A., AlGhafees, M.: Gap analysis between state of practice and state of art practices in agile software development. In: 2015 Agile Conference (AGILE), pp. 102-106. IEEE, National Harbor, MD, USA (2015).
11. Azman, N.: The development of IoT tele-insomnia framework to monitor sleep disorder. International Journal of Advanced Trends in Computer Science and Engineering 8(6), 2831–2839 (2019).
12. Schuh, G., Rebentisch, E., Riesener, M., Diels, F., Dolle, C., Eich, S.: Agile-waterfall hybrid product development in the manufacturing industry–Introducing guidelines for implementation of parallel use of the two models. In: 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), pp. 725–729. IEEE, Singapore (2017).
13. Abdelghany, A., Darwish, N., Hefni, H.: An agile methodology for ontology development. International Journal of Intelligent Engineering and Systems.12(2), 170–181 (2019).
14. Aydemir, F., Dalpiaz, F.: A roadmap for ethics–aware software engineering. In: Proceedings of the International Workshop on Software Fairness, pp. 15–21. Association for Computing Machinery, New York, NY, USA (2018).
15. Zarour, M., Alenezi, M., Alsarayrah, K.: Software security specifications and design. In: Proceedings of the Evaluation and Assessment in Software Engineering, pp.451–456. Association for Computing Machinery, New York, NY, USA (2020).
16. Aljawarneh, S., Yassein, M.: A conceptual security framework for cloud computing issues. Cyber Security and Threats, IGI Global 12(2), 12–24 (2018).
17. TCSEC - Glossary | CSRC, https://csrc.nist.gov/glossary/term/TCSEC, last accessed 2021/01/02.
18. Sterne, D.: On the buzzword 'security policy'. In: Proceedings of IIEEE Computer Society Symposium on Research in Security and Privacy, IEEE, pp. 219–230. Oakland, CA, USA (1991).

19. ISO/IEC 9126-1:2001, https://www.iso.org/standard/22749.html, last accessed 2021/01/06.
20. Merritt, K., Zhao, S.: An investigation of what factors determine the way in which customer satisfaction is increased through omni-channel marketing in retail. Administrative Sciences 10(4), 01–24 (2020).
21. Merritt, K., Zhao, S.: An innovative reflection based on critically applying UX design principles. Journal of Open Innovation: Technology, Market, and Complexity. 7(2), 1–12 (2021).