9 - Evaluation

Contents

9.1 The Project's Tools and Language

9.11 Effectiveness of the Programming Language

Overall Java was an excellent programming language to have used for the system. Before I go into detail about the experience I had with the language, **I generally feel that the tools and functionality available allowed the project to meet its potential**. However, this was not just down to the language being 'easy' or 'simple' to use, as on the contrary I feel that it's quite the opposite. I believe the language proved to be advantageous solely due to me taking my time before prototyping to understand the fundamental aspects I would need to incorporate.

Other Languages

As briefly mentioned in the investigation, Java was the best language to proceed with due to it being the most adaptive and widespread language at my disposal, while other languages like **Microsoft's C** or **PHP** being noteworthy options to choose from, I ultimately went with Java due to my **existing familiarity** with it and the **available support around me** made for an easy option to choose. Overall I believe I have made the correct choice as I feel the product has been produced with a high degree of quality and **robustness**. **Java's English like structure** and **high level nature** made it appealing for consideration and generally **more favourable over other examples** like C, which in particular focuses on **a more procedural top down approach to programming**, by having an object oriented approach we can utilise this when creating major entities for the system. To add to this Java makes use of **error catching**, this was critical for the validation of the system and would have resulted in drastic changes if unable to have seen this implemented. Finally Java utilises the option of **multithreading**, this meant while the program was executing it was possible for new threads to be ran, while this wasn't found until later it was nice to know it was an option. This was a common issue throughout Java, it provided many tools for the users disposal but they were never found until much later on into development when I could no longer make full use of them.

External packages

While certain packages were available to incorporate into development such as **Java.SQL** to replace data manipulation with querying functions and **Java.Crypto** for encryption. I generally felt that working from the ground up with custom source code allowed me to **naturally suit the program to the requirements needed** and **provided a sense of control** with how problems were going to be handled. To add to this, another drawback with the inclusion of such classes was that I had no experience whatsoever with them and meant I would be relying on a feature I had **no familiarity with** to perform crucial tasks. Because of this I went and established everything unassisted, for instance with querying databases I could have relied on method calls like **Statement.executeQuerey("SELECT * FROM Admissions")** but these were overlooked instead for custom searching algorithms like binary and linear searches. While overall having the chance to have used these features would have probably improved the system vastly, regardless **I feel minimising external package usage was the best option to have gone with**.

Object orientated approach

What worked in my favour with Java was its use of an **object oriented paradigm** and associative **object hierarchy**. As I knew about this prior to the design of the system, I made sure the code would draw greatly from its core aspects. This meant that I could drastically reduce attributes needed and method lengths for classes through **heavy use of inheritance**. This resulted in the code using a more **modular approach between these entity classes**. A big accomplishment of using Java for the project was the use of **polymorphism** to overwrite and create custom event listeners. By adapting existing methods in the Java library, a process that took many hours to grasp but is now common throughout the system, I ultimately reaped the benefits from by including features that would have been unavailable to incorporate without the use of **custom listeners and events**.

The complier

The language also proved to be effective through the use of the **compiler and associative Java virtual machine**, this essentially allowed the source code to be **system independent** and only relying on the compiler and virtual machine being installed on any given device. This offered lots of portability for distribution of the system. As Java uses a complier for translation it presented a wide range of **functionality for portabilit**y and **future packaging**. While it was never needed the option to run the source code on different machines was always available and meant that if I ever needed to work from a different device I was never restricted to certain operating systems or hardware. This means for the actual submission of the software; it can just be zipped up and sent to the moderator who can just compile and execute it without needing to worry about the correct version of the compiler being installed.

Extending and importing Classes and Packages

Another key characteristic of development with Java was with the ability to import existing packages and classes. Two of these native classes was **Java.Swing** and **Java.AWT** both of which were extensions of Java already, this allowed the creation of graphical objects that were interactable with the user. Swing proved to be crucial to development due to the systems **heavy reliance on user interactions with the interface**. As the package had also utilised an object hierarchy it meant that components could call upon the same methods as others if they shared the same parent class, allowing for clever use of method calling to all components. This overall proved to be essential when dealing with many different objects that all required to perform the same action.

However the package was not without its faults, **a point of contention was the look and feel of its available components**, when initialised, all objects had the same dimensions and colour scheme. This meant that a lot of time was spent setting up colours and formatting items. To add to this an advanced aspect of Swing was **its utilisation of layouts and positioning**. While it would have been effective to have included more of these, like **CardLayout** or **BorderLayout** for instance, my early experience was primarily with setting the panels as **null** and manually positioning every item, a process that took up a fair amount of time and resulted in unscalable windows, a feature I ultimately regret not changing from the outset.

9.12 Justification of Tools and Techniques

IDEs and Text Editors

While its more common to see integrated development environments like **Eclipse** and **NetBeans** used to create large systems, I opted to use a more simplistic text editor to produce the system for various reasons. While the editor lacked more advanced functionality like **code optimisatio**n and **error diagnostics**, I felt confident enough to use the **NotePad++** application to develop the program, due to my familiarity with the editor. What ultimately kept me using the program was my now heavy reliance on three aiding features, the **find and replace tools** and the **ability to collapse brackets**. As it now stands the number of lines of code alone for the Gui now surpasses 11000, it would have taken too long to isolate bugs and find certain parts in short periods of time, the aforementioned tools **greatly reduced this** and meant I could get on with development. With the ability to collapse large portions of the system it allowed me to have desired methods visible at any point of time, this made finding the sections I was working on **almost instantaneous**, this proved to be time effective when working with a project of this calibre. To add to this, another feature was the ability to split the window into two resizable sections for multiple class files, this greatly aided development as it allowed the viewing of multiple files without the requirement of another monitor or opening a new instance of Notepad++.

Despite this, the editor wasn't just a word processor. Some aspects of typical IDEs where provided by the program. The most beneficial of them being the **Highlighted syntax** and **statement suggestion**. These tools aided development as it reduced the chance for grammatical and syntax errors throughout. For instance when I needed to call an item that had many characters to enter I.e. "**SystemAdmissionList[currentAdmissionIndex]"** it would suggest possible words as I kept entering and allowed for the system to avoid incorrect variable names. To add to this, colour coded syntax assisted development due to the fact when I created a new statement it would highlight when a bracket hadn't been closed by creating a red line, while this feature wasn't used too frequently, however during lengthy logical statements, **"if((Character.isLetter(c)==false)&&(Character.isWhitespace(c)==false)&&(Character.isDigit(c)==false))"**, it proved helpful to see when a syntax error had been made.

Outside community and online Documentation

A large draw to Java was also its **outside community and online documentation**. **Stacked overflow** proved to be a pivotal website that resolved many of the more complex and demanding problems I faced. As Java has many intricacies to it, the guidance I found on Stacked overflow allowed major issues to be resolved. One of these was the constant battle I had with the formatting of components with the **Java.Swing** package and the **Java.Graphics** class, having the ability to look online and see many different examples was very useful in overcoming unique problems. Despite this a constant issue I had with the website was the answers provided by users **being far too advanced and way above my comprehension**, it would be clear on inspection the sections I had written and sections that I had just copied. As a result I only drew inspiration from examples I knew I could explain to others or could debug if errors were to appear. When stacked overflow did fail on me however, I always had another website to fall back on, the **Java Api**. While these collection of websites contained **much more technical information** than I needed, it often showed me all the available methods to an object, this was a godsend for Swing as it meant I could **see the options available to relevant subclasses**, however this was usually a last resort as the majority of information was **overwhelming**. Despite this, what I could understand meant that I had access to information that had greatly aided development and was greatly appreciated.

Panel navigation

While Swing was great with **initialising components** and assigning them to other objects, a process that took many iterations and attempts to perfect was **navigation of panels**, this is due to the **lacklustre built in support** provided by Swing other than the bland **JTabbedPane** component provided by the **awt.Container** class. As it was the convention for other students to just discard the panel and re-initialise them when needed, overall a demanding process for more complex panels. My alternative was to instead keep track of which panels had been generated and **then keep the order of panels in a stack**. This also allowed for the rolling back and returning of panels along with the existing data, this **proved to be popular with many users** during testing. So while I was happy I reached a viable technique, I was just frustrated nothing like this was in place already and I had to reach a solution on my own.

File manipulation

As data needed a permanent location for the system to run, **file manipulation acted as an underlying backbone to the applications operation**. However a major flaw to Java was its lack of functionality with its file **buffered readers and writers**. As a result **custom methods where setup** to compensate for the absent processes. As the writers could only read one line at a time in a linear fashion it meant that **accessing isolated lines became a strain** . This instead resulted in **independent methods** being initialised in the **User** class to allow the system to read and write certain lines to file. As a result this created a b for the system and has resulted in a **fundamental limitation** for performance, where over time as more and more items get added to a user's file access times increase linearly on all cases as regardless where the desired line is the entire file is read.

Personal documentation

As a core requirement of the project, strong commenting was a necessity, I approached the task in two very distinct directions. Initially in the prototype, due to the systems much smaller line count I went **line by line** and **manually commented every line** after finishing the program. This **proved too much for me** at the end and as a result I planned to tackle documentation in a more effective way. When it came to the actual development, after the painful memories of prototyping I carried out this plan and **commented by typing a list of bullet points for every method followed by block comments at the top of each section describing purpose and functionality**. This was by far the more sustainable and effective approach, with over 250 different method calls and classes it was clearly the **better call of judgement** to have gone with. To add to this, planning beforehand also allowed a sense of cohesion to what I was doing and allowed me to **poke holes through weak approaches and get a sense of how a task would occur.**

9.2 Similar Commercial Systems

CareCloud

(Software discussed during investigation)

To start I feel the best software to initially assess would be CareCloud. I largely regard the EHR (Electronic HealthCare Record System) as a major influence in the early days of the project due to it establishing core features and actually inspiring certain aspects like the **contact bar and navigation system** and as a result I have no need to go into detail on the exact features as they are already present in the my project. Overall I feel that I have done well to follow in the systems footsteps while making the project feel my own and not too much like an imitator despite **the common similarities**.

CareCloud provides a single integrated financial, clinical and administrative set of tools to the userbase and allows for a plethora of features to create an overall seamless experience for all users. The software relies on: **Patient Engagement, Security, A modern desk experience** and allows for but limited to: **A comprehensive drag and drop Schedular, Managing patient Demographic information, Fool proof billing and Data analytics**. All of which are integrated into a single service for potential customers to purchase using a monthly subscription model. My software compares to this by providing a similar option and allows for a standard EHR experience without any of the more unnecessary aspects like allowing for the personalisation of the interface. Because of this, my system seems to have much less to offer, however what I believe it does is perform exceptionally well for a system I can guarantee has had significantly less time in development.

Similarities

Where my system draws the most similarities with CareCloud would obviously be in **the updating of patient demographic information** such as first names and addresses, however, CareCloud takes things one step further in the classification and **inclusion** of more **important information** like **vitals** and **immunizations**. On my part, fields like these were **overlooked** in **design** and meant on their discovery much later into development it was **too late to incorporate**. For the second feature, with my **system it is possible for patients to establish bookings** with their consultant for a unique time and location in the hospital through selecting items from drop down boxes and allows for possible cancellations and updates. For the other system, **scheduling appointments was similar** in the respect to this but was in much greater detail as you would assume, this came about in the form of **dragging and dropping** patients into time slots. I feel my features allow for a more **simplistic approach** to the task but ultimately, CareCloud's advanced functionality clearly makes it the **superior option**, however we can clearly place this as a limitation of my system due to my **lack of experience** using dragging features. Another key similarity between the systems was that **both programs allowed for the use of templates to reduce work**, in my system, **text based** documents came with pre-written data to help **reduce** entering **unnecessary information**, similarly with CareCloud, the program allowed for virtually all documents to be automated through a merging system relying on user habits analysed throughout the system. Finally the last similarity would be in **the use of reminders and notifications**. In my system when users have information edited or appointments amended the action gets attached to the patients account for them to see and eventually delete once read. CareCloud approaches this likewise in the form of "**Notes**" when loading an account a **popup is generated** displaying **relevant** information for the **consultant to see**. While I like the idea of a **broader notification** system like CareCloud's approach, I feel that my method allows for the notifications to be kept for whatever reason and allow for a more **sophisticated** feature you would see on **widespread software**, to add to this it also follows the app like interface that has been a constant inspiration throughout the system.

Differences

While **overlooked** in the **investigation** slightly, CareCloud generally is **aimed towards smaller** independent medical practices like **local clinics** and tries to provide an **all in one solution**, whereas my project revolves around a **larger establishment**, Euxton Hall Hospital and aims to **resolve** the extensive **archiving** problem currently present. As you can see both systems have two different main goals they attend to accomplish. Regardless CareCloud's isolated features seem **interchangeable** with any other EHR and we can generally see this from the resultant product created and so there should therefore be no reason to disregard the disparities. A key difference I would compare between the two systems would be my **lack of any financial aspects to the program**, like mentioned countless times before this would have ultimately been **out of scope** and has resulted in the **monetary processes** remaining **mutual** to the programs operation at Euxton. Because of this we can refer back to the **original limitations** of the new system stating the project would **not** be **able to incorporate** every aspect present at Euxton and instead should primarily **focus on the core issue affecting the system**. Even more, another difference worth mentioning would be the "Built in diagnosis lookup" CareCloud boasts in the key features segment of the website. Again this was **another limitation** of my system that initially saw a **more dilute aspect included** but eventually resulted in **omission of the idea** due to **many reasons** I'll explain later in section 9.4.

Another difference with my system and the commercial software is that with CareCloud consultants have the ability to **access recent patients**, while mine **used to have that feature** it was eventually **phased out due to redesigns** as a result of the feature **taking up too much space** on the homepage. While both programs utilised appointment notifications, we **approached them differently**. With mine, the alert was held on the **home screen** with a button that allowed **for direct access to further information**, whereas with CareCloud external communications have been set up to allow **for SMS messages and emails to occur**, again this was a major **limitation** of my system as **no external infrastructure** was going to be utilised, this resulted in a restriction on what was possible for how the patient was going to receive the information, ultimately I believe **the best approach was used for what was available.**

Interfaces and hardware

The final aspect of CareCloud worth mentioning would be of its **interfacing and accompanying hardware**. While my program is designed to be ran on **Java compatible devices offline only**, CareCloud bolsters over **three different platforms** to provide their service including: online **Websites**, **mobile applications and kiosks**, provided by a partnered hardware supplier **First Data®.** Other comparisons would be from both systems **modular 'app like'** interface providing an experience a user **may feel familiar with.** Despite this CareCloud again **surpasses mine** by allowing for a **configurable experience** to **suit the needs** of each practise separately. Whereas my project uses a **static layout** preventing any further changes outside amendments to source code. However due to my heavy use of the **BoxLayout** I have been able to see **some formatting of items** and configuring the contact box depending on the type of user; this could **potentially see further development** in future iterations.

Praxis

(Software discussed during investigation)

The next piece of software I will discuss will be the other system researched during the investigation. Praxis claims not just to be an EHR but rather **a template free medication tool** that uses an artificial intelligence called **'Concept Processing'** that **learns user interactions** providing smarter results for the client. The developers of the software claim for it **to reduce the time spent performing administrative tasks** and **allows users to practise medicine 'their way'.** They also claim that changeover can be an **effortless** process through an **automated migration system.** As this system was looked into early on we can see **many of features present throughout my program** and as a result can compare many similarities.

Similarities

While CareCloud **inspired the main 'look and feel'** of the system along with some minor features, I feel that nearly all of the underlying aspects of my project where **directly influenced** by Praxis' main functionality. A core aspect of Praxis I imitated was the **software's minimalistic data entry concept**. Where the program would **automate monotonous tasks** for the user which essentially **reduced workload.** For my project, a main **limitation prevented this from occurring on a much wider scale,** this was down to certain **sections being compulsory** like creating accounts. This was mainly because they **required lots of information to be entered.** Instead my interpretation would directly **address sections where data entry was unnecessary and required large portions of time** that may be used many times like **document creation** or **appointment booking.** This limitation would also see more advanced features like the diagnosis generator kept as a **manual process during later stages of development.** Overall I feel that incorporating this into the system in smaller sections allowed for a **much larger control over broader aspects** and meant that **no feature felt overlooked or basic.**

Another similarity would be the **ability to scan documents** and **creating a virtual copy**. Both mine and Praxis utilise a changeover feature that allows the user to take **physical assets and produce high quality images of them,** this would eventually move to the document being attached to the relevant account. However **we approach this very differently.** With my attempt users will **employ a mutual scanning device** to create a **high resolution image file** which is then stored on the hard drive, after this the user should enter the system and follow the process to create a legacy document. Unfortunately **a key limitation** to this is that the **obtaining of digital files** is a **process** that **can't be directly provided** by my system as **special hardware would be needed** to perform the job, as a result half of the process has to remain mutual. With Praxis', the software relies upon a **partnered external** company **Scanaway©** to do the heavy lifting and allows for batches of old documents to be scanned in **large quantities.** For everyday performance I feel **mine** appears to be the **better solution** as it would be **faster** to perform on **isolated documents,** while initially Scanaway would clearly possess the advantage **due to its mass fabrication of the scanned documents.**

Differences

Despite Praxis having some large similarities with the system **not everything was the same**. For instance, with Praxis the system overall seems to primarily **focus** on the main use of the **Artificial Intelligence 'Concept Processing'** to minimise the work required. My software differs from this by instead relying on **logical conditions** to trigger automation **when necessary.** However a **limitation** of the system was always going to be that the **majority of data entry needed manual input** and as a result the ability to utilise such **advanced features was always going** to be **minimal.** To add to this, another key difference would be Praxis' **lack of any patient functionality,** while I try to provide a somewhat **full experience for the patient,** allowing for a versatile range of **tools and features,** Praxis has **nothing in place** and is entirely a **staff orientated service.** Finally the last aspect apparent was that once again Praxis, like CareCloud, **provides a large focus on financial and business analytics,** a service that was **omitted early on in the investigation** as there was no need to integrate a working and **competent service.** Overall I feel most users would agree that for Euxton, the hospital would much rather benefit from a system like mine over a program that tries to **perform everything with varying degrees of success.**

Interface and hardware

Here the interface of Praxis provides a **serious and effective display** to the user, from the evidence provided throughout the website and online training videos, the software presents itself as a **formal medical tool** to help provide a **productive service to the user**, despite this, the **general look and feel** seems a **little outdated** with a **blander and much older feel compared to CareCloud**. However despite my system aiming towards a more **contemporary and an 'app like' interface**, I feel the minor **lack of colour** just prevents this appearance from being reached and as a result **follows more towards looking like Praxis**. However I generally feel both systems have been **tailored to suit their respective needs**. For instance as mine has both patients and employees, a compromise had been created where both **speed and grac**e has tried to be **established**, whereas with Praxis we can clearly see **efficiency is the main goal**. As for hardware once again **my system comes up weaker**. While my system is bounded to devices capable of **running Java offline**, Praxis **provides instant access** on any computer, tablet, laptop or any handheld device **through a cloud based service**. This ultimately allows for instant portability, whereas mine is bounded to a limited number of devices all of which must allow for the storage of **not just the software** but the database as well, again **another clear limitation**.

DrChrono

(Highest rated EHR on https://www.softwareadvice.com/medical/)

While it's important to compare my project with the key influences investigated early on, I think it's **more valuable** I assess my program with new and **different commercial software available**. Because of this, the final system worth discussing is DrChrono an American based EHR. Overall DrChrono provides a **fully customisable EHR** to suit a practise's needs by providing the **standard features for consultants to use**. The developers claim that **mobility has always been a central influence** of the system and cite that **all aspects revolve around the user** experience. Because of this I chose the system due to both mine and said programs **shared fundamental idea** that efficiency should be a main focus, however we achieve this in varying ways. The program provides many services like: **A mobile EHR**, **Revenue Cycle Management**, **Practise Management and Medical Billing software.** Whereas my project provides a **sole focus on the organisation** of documents along with some minor focus on **appointment management** and the **patient demographic**, again Euxton **isn't currently looking into** replacing **financial aspects** of the system so such inclusion of the feature would be **unnecessary and still remains mutual to the program.**

Similarities

A large similarity both systems share would be the inclusion of **patient access to the system**, unlike Praxis, DrChrono provides **patient interactions** with the system through the use of the **subsidiary program OnPatient**, this allows for patients to access all suitable information like **medical records and insurance information**. Likewise my system allows for patients to have an **unprohibited view** of **all of their respective data**, while some of their information **won't be editable to them**, they can still inspect the fields. As a result I feel that having a **more open approach** with patients **results in better compliance with data protection**. Another similar aspect would be that both systems use **template creation**. With my system, users can create documents using pre-made templates which they can edit freely. With DrChrono the software allows the creation through **dragging and dropping of individual components**, similar to the tools provided by **Muqups**. With this feature I feel my program, whilst **a slower process**, overall provides more freedom, this being due to the text being **limited to certain characters** unlike being forced into using **dropdown menus and tick boxes**. The last key similarity would be **security**. With DrChrono, the system complies with **HIPPA**, an American law surrounding **access to private medical information** and ensures that data is **encrypted** and **inaccessible** to those with **unauthorised access**. My system illustrates similar principles by preventing information from being accessed by consultants or staff who don't meet the **requirements to view the fields**. Likewise all data on the system is always encrypted to ensure data protection. However as my choice of encryption being **symmetric** and a novice choice, a Caesar Cipher, it ultimately means that the data would never truly be safe and as a result is inferior to DrChrono in terms of security.

Differences

A key difference that sets aside the program from other EHRs is that DrChrono allows for patients to attach child accounts on the system and **allows for family monitoring**. My system is yet to include any **subcategories of the patient entity** and so for now the feature is **not present**. This leads onto a key limitation being that the patient class acted as a **generalised user for all patients** and meant that nothing to **cater for different users was established**, as a result while my system aimed for **efficiency it ultimately came at a cost of usability** for those with disabilities and users unable to use a keyboard and mouse. As you could imagine for a hospital this is **anything other than ideal**. Another key difference is the utilisation of a **messaging service to allow for communications between users**. DrChrono allows for the use of built in messaging allowing patients to **directly contact** their consultant on the system. Again this aspect has been **left mutual on my system** as all communications are established by **direct contact** or by **one way letters** sent to the residents house, this has ultimately been left due to the **lack of infrastructure to accomplish such a demanding task.** Another unique difference is that DrChrono allows for the use of **sharing educational material**, while my system allows for the searching for medical terminology, it **pales in comparison** with the ability to **distribute medical information for the patients**. Finally the last key difference would be the **direct integration of the labs aspect and accompanying image requesting features**. This allows the consultant to directly request and access laboratory issued results and images. While I do have an option for consultant test results to be generated as a document for the patient, this aspect is the lab distributing results to the consultant. **This feature would have to remain mutual in my system regardless** due to the systems **lack of any association with imaging laboratories**.

Interface and hardware

A key point to discuss would be the use of the **interface and accompanying hardware.** With my system **I utilise many buttons and scroll bars** to allow for a **dynamic and constantly altering user experience** to allow for **possible changes that would occur** during use of the system, DrChrono follows a similar idea by allowing for **scrollable windows and uses webpages** rather than custom software **more associated with typical EHRs**. Overall I feel the use of **Swing works well** for a system like mine as the project has **limited features** and having the system run **offline reduces the risk of cyber-attacks**. However we can see **a clear difference in the use of hardware**, while my program uses **Java** and is therefore **limited to a number of devices.** DrChrono has a heavy focus **on mobile hardware**. Due to the use of handheld devices, a unique feature the system includes is the use of drawings and annotations using **Apple products as an official hardware supplier**. I feel for this aspect, while certainly an **impressive feat to integrate**, such **unique ideas** for the system reduce the overall simplicity and only complicate things further, an aspect I was always wanted to limit. To add to this the DrChrono allows for **full incorporation of the Apple ecosystem** allowing for **direct access** to the **Apple Health app** located on devices such as the Apple watch. Again while for other customers this may be **a reliable way to obtain data**, for a hospital that needs to virtualise documents it's completely **unnecessary** and would ultimately **cost the Hospital money** that **could be put to better use**. As a result I feel my **solution provides a more elegant result** by providing features tailored to the issues at hand and provides processes that are only needed.

9.3 Good Features and Shortcomings

9.31 Good Features and Shortcomings

Requirements and Performance

To begin, I believe that overall the system **meets the functionality** set out in the investigation and **achieves the goal I wanted to accomplish,** to provide a **virtual alternative** to the filing process at Euxton Hospital. I can confidently assume this due to my **extensive alpha and beta testing** performed on the program. I feel the system provides the **necessary processes to achieve this** and performs them in a **suitable time scale.** However like any other system, it's clearly **not without its flaws.** As time was a constant issue with **the project** it **has** resulted in **many imperfections,** like for instance, the **limited number of document types** or the omission of the printing of results. Despite this, I feel they **don't diminish** the user experience in any way, **but rather are just aspects waiting to be fixed or established later on.**

A strong characteristic of my system is that the majority of features have been included with **a high degree of functionality** and that **no aspect feels it was introduced at the last minute,** despite certain objectives like encryption being implemented only a few days before submission, I'll talk about this later in 9.4. Generally, I feel that **every process feels worthy enough to be included on the system,** this is due to all the attempts I made to get the objectives perfect during the investigation. By having **time to consider each objective** it meant that **unnecessary features were avoided,** likewise, by having the time to prototype it **allowed for refinements to be made when needed.**

In terms of Performance I believe the system is again **more than practical to see actual use at Euxton.** While development resulted in **less than optimal design choices,** I know the system is still more than **capable of withstanding larger workloads.** However a key weakness of the system was down to the **searching methods I had incorporated.** Despite having used **binary searches** for every situation on the prototype, it's only now **used for certain aspects.** This is because of a key criticism from the prototype that was testers **disliked having to enter the entire primary key for a single result,** instead the process was **drastically changed** to allow for the searching of any item, once queried an array of matching items were returned to the user. This **required a linear search** having to be used to ensure all items were compared, however this **came at a cost with an increase in time spent searching the list.** Regardless after testing the search using an array of over 1000 items, it is **clear that the process remains virtually unaltered in real time,** only with much larger data sizes will I see a reduction in performance, however it is unlikely that a single consultant has over 1000 patients so **I don't find it an issue worth exploring further.**

To add to this, while initially I believed **hardware** played a large **contributing factor for the reliable performance I was experiencing** on the system, after months of having a perpetual fear that it wouldn't be runnable on any other device **I was very surprised to have the system run well on college hardware on the first attempt.** I think a large contributing factor to this and o**verall strength of the system** being the use of **local variables and the panel management of the system.** After some slight investigation early on with my experience with Swing I noticed that the main reason performance was so lacking was **due to the constant re-initialising of graphical components.** As a result I made sure from the outset I would get it right from the start to prevent this from ever occurring on the program. Having the ability to initialise **Swing components once has easily become one of the best aspects of the system,** despite taking a considerable amount of time to perfect. We can see the idea clearly works. This can be supported by the **testing performed throughout the development** and the **navigation testing towards the end of project.**

Usability and Robustness

With usability I feel that the system has both major **strengths and weaknesses associated** with it. The system does well to **provide a basic set of features** when a new account is created to prevent overwhelming a user. As a result it **allows for the user to get acquainted with what the system has to offer.** This is a result of the **dynamic object creation** that is common throughout the program and allows for new Swing components to be created when needed. For instance when **a new admission or document is attached to an account** an accompanying button is created to allow direct access to it. This overtime **will allow the user to get further functionality** as they get accustomed to the program. However a downside to the system which affects usability is that **no on hand guidance** is in place for new users, so **while users aren't overwhelmed** with what they can see, without my help **they wouldn't know how to use the system** particularly well. This could be because the design might be regarded as **nonintuitive in terms of the layout**, regardless it is **an aspect I would like to see resolved at some point in the future.**

Another good aspect for usability of my **project is validation** and **data entr**y, due to the strong **validation testing** and **robust design** of how the system deals with input from the user, the system manages to deal with **erroneous data very well**. I can confidently say that due to the extensive attempts to test every field on the system. For proof we can call upon my **validation testing in the last document** and as a result we can account for **every type of invalid character or datatype imaginable** to the user. However while the system is **good at detecting these invalid values it's very ineffective** with how the results are shown to the user. For example when a user was **creating an account** and enters **two wrong values only** one value was informed to be invalid, as you can see it is **withholding information** the user should be told. With how the validation method works it would have **been too much to overhaul the entire process in a few days** and because of this it was left in.

Moving beyond usability **another key aspect** of the **system was robustness**. Again, **while I'm not saying the system is perfect**, I strongly believe after much extensive alpha and beta testing the **system is in a very strong position and feels almost bug free**. The main reason I feel this is the case is because **the errors that do occur are solely visual** and **cause no problems to the files** of the system. This due to two main reasons. Number 1, the **system only every updates account instances from file information** and the new data generated, **never from the current instance held in memory**, when **an item is being written to file** rather than using the instance to write data to file the system instead rereads the file and **makes the necessary changes using the correct data**. This ultimately prevents any **discrepancies from occurring**. Number 2, if any file corruptions were to occur they would be **isolated to that account only**. This a **major strength with the new redesigned files** being user independent on the system. While unfortunately a user is still unable to access the account the system can still be functional. **This was a necessity following the fiasco that was prototyping**. If it's not obvious yet, during the prototype, when an error occurred in the central file, **the entire system ended up being unusable**. As a result this is what **ultimately prompted the overhaul we saw** introduced during the post prototype as it would **mediate much of the issues I had with debugging**.

Major sections

Patients

To summarise, I feel that the **patient aspect of the system is one of the strongest parts of the program.** It offers the necessary features to patients and **provides a simple initial interface** but allows for **advanced processes to manage their admissions and accompanying documents.** Even more another strength for this entity was its **object hierarchy,** despite the reality of the patient having many admissions, if we were to have used the **child class of admission like a normal object only one instance would be available to the patient.** This quickly became an important issue to fix otherwise I would be **unable to use the class hierarchy how I wanted to.** Because of this, it is clear that another technique was used to obtain the many instances. In my approach we declare **an array of admission classes as an attribute to the object.** This ultimately **allows inheritance to still occur but we can still have more than one instance of admission available to the patient.** To add to this, it also means we can have an effective data structure to handle the many objects.

However an aspect I felt seemed weak was the **entity's accommodation for disabled users.** While I do store information regarding physical disabilities **I do nothing to help cater for them.** While I would have liked to have seen some help for them **it was always going to be down to time and resources that prohibited me from doing so.** Another great aspect I loved however would be the **admission creation system.** I feel the **human silhouette and highlighted sections was a great inclusion** and added a more **professional feel** and also **helped aid users who may be confused** from just the descriptions. The final aspect about the patient I thought was great was the **direct access to legacy and new documents.** By having both types in one section it has **allowed for both to be easily available and prevents any confusing situations from arising.**

Consultants

Again, this is **another strong entity on the system.** While with patients the users were all assumed to be unfamiliar with the system, consultants **would be used to working with the program on a daily basis** and so I made sure that it **was designed to be as effective as possible.** Overall I feel the program **shows this very well.** For instance a great example of this would be the ability **to view patient information despite not being logged into the patient's account,** allowing for easy retrieval of information. Another good example would be the **creation of admissions from a consultant account.** While it was an aspect that was nearly excluded for being too much to fully implement, it was left in and **allowed for a great aspect to be included and came at a minor cost of time.** The last excellent feature worth mentioning would be the **appointment calendar present on the system.** By having a personal list of appointments on the consultants' file the system **allowed for double bookings to be completely avoidable** and meant that **no errors could be made by patients selecting invalid time slots.**

However, the section **was defiantly not without its faults.** An aspect I was disappointed by was the **bookings on the consultant homepage** when the user selected the desired appointment they wanted to inspect. Instead of being moved to the actual admission booking **a basic panel was initialised containing the information about the appointment but was unable to edited.** As you could see this would be **incredibly annoying to try and see information regarding the related admission** or if you wanted to update the booking information, having to instead locate not just the patient but the admission as well. Despite this **I do love the scrollable time slots** and really like **how the buttons work from a functional standpoint** using a hashing algorithm to locate the index. Finally the last aspect that I feel is a weak part to the consultant is **the location of the new terms button for the entity.** While slightly a minor issue **I don't feel comfortable where it was positioned** and feel it could have **been better located on the contact bar or on the glossary panel itself** as its current location feels completely out of place and looks like it was put there at the last minute.

Staff

Overall while **I never wanted this to occur** I feel that **despite all my attempts** to give meaning to this entity **there is still no main reason to keep the user on the system.** I think this is the case due to the fact that after using the entity during testing I **always ended up wanting to do everything from a consultant or patient account** and never a staff one. I believe that there is **no main reason that the account needs to be kept in the long term** and is **overall a crutch to the system** as besides the legacy document creation they perform, they **provide no major use that distinguishes themselves** from consultants. However that being said, **I feel as a singular entity the account is perfect** as the evidence provided from Mrs Suzanne Tomkins during testing shows us, she clearly found the system **easy to use and was effective being a staff account on the program.** However despite how well Mrs Tomkins felt using the system, **I still feel disappointed with how it turned out.** For instance, the **homepage feels uninteresting and could have literally been a search bar** directing users to the correct account. I feel if I was to introduce a new set of features, for example, **business analytics and finance**, which would be an amazing set of features to include, **the entity would have had much larger significance on the system.** However, as the project already has over 50 objectives this would have been **an impossible challenge.**

The aforementioned legacy document creation is an **excellent feature I had included.** The ability to select an image and then see it appear in front of them **in real time** was an **excellent aspect of the staff account,** even more the fact that all images are then copied and written to a new location is a **really advanced section that no one will appreciate due to it being a backend feature.** However, I feel despite being an amazing aspect to the staff account **it isn't enough to keep the account feeling worthwhile overall** and could **easily be moved to the consultant entity if needed.** Finally, the last gripe I have about the staff section is **restriction to data.** While personally I felt that all the patient's information should be visible to the staff account, **testers during the prototype thought otherwise** and so a compromise had been reached were some data was visible but the rest wasn't. The result of this has **left the staff account feeling in a limbo state of usefulness.** While they can see the patient demographic and admission information they are unable to see the documents and **as a result of this no party feels satisfied with the solution.** I feel that **a rethink is in order** for this entity to meet its potential.

Management

While not the best section of the system it is certainly not the weakest. The **management entity does have many great aspects that I am proud of,** however, there are a few issues I have with the entity I feel **prohibit it from reaching a state I'm completely satisfied with.** For instance the **interface,** after hours of **contemplating the user interface** I believe that using the command line **was both a good and bad thing to have chosen.** I feel it was beneficial as it made th**e process of creating employee accounts to be a quick and painless process.** To add to this, other aspects I set out to include, like the **employee archiving,** would have made a **tiresome task to design suitable interfaces for.** Having them text based seemed the **most appropriate thing to have done in my situation.** However, the fact it is text based makes it **really unappealing to the user and as a result made the system look unprofessional and even incomplete.** Despite this a **great aspect would be the audit log.** I'm proud of this section as most of the actual work **was taken directly from the prototype due to how robust the code was.** I **really like the searching and refining tools available** to isolate set time periods as this allowed for a **more effective way of helping users find what they are looking for.** Even more, I am also pleased with how the **table came out in the end despite being a graphical interface,** the work that went into **pausing the thread while the graphical component was running was a brilliant aspect to get working flawlessly in the end.**

The final and most noteworthy aspect of the management entity was **the archiving system.** I believe this to be the **hardest method I have ever had to write** but as a result has become the most complex and **brilliant method of the system.** Having the ability to archive an employee is on its own **basic,** as it is just the **updating of a boolean value,** however, when you consider that active patient accounts are still attached to an archived employee **we then see a major dilemma**. We need to **disable access to the account but also move the patients to another active staff account.** This then becomes a further issue **as not only do you have to distribute the remaining patients** between the available staff but then **update both the patient and new staff account** to link them together. This aspect took a very **long amount of time to get working in the end** but this was without a doubt **time well spent,** despite the fact the feature will certainly be **overlooked**. Overall despite the entity **not looking perfect yet,** I feel the **functionality is more than ready to see use** at Euxton. For me the management entity is a **hidden gem for the system** and **contains features that highlight how through the system is at the moment.**

Missing objectives

Here I now get the chance to **fully explore why these objectives were excluded from the system during development.** While others like Objectives **number 31 and 32** were omitted because of **changes to the system resulting in them no longer being required** the objectives below were always planned to be **included but for some reason they were left out at the at the last minute.** Now I get the chance to justify the decision to remove them.

Determining consultants and diagnosis

This objective was by far **the bane of my development**. To put it simply**, the idea was too advanced to create a feature that was safety critical**. Even more, the aspect **had not only been left far too late** into development to start work on but **rather the approach I planned on undertaking would take up too much time to develop** and would **require medical knowledge I never possessed**. Overall this accumulated to a feature **I knew for certain I wasn't going to be able to pull off to a passable standard along with the rest of the system.** Looking back at it now the idea now **seems ludicrous** that such an advanced feature like this **would take such a minor role in the system.** It was absolutely the **best choice to have excluded a feature like this,** to be honest it **should have been omitted during prototyping** but was left in due to my **arrogance** thinking I had enough time to fit it in during actual development**. Nevertheless the right call was eventually made** as it is almost a project in itself.

Printing documents

While **clearly the ability to print documents was never going to be a vital solution** to the archiving problem at Euxton I **ultimately feel that this was a missed opportunity for a unique feature** with **real life applications** to have been incorporated. While regardless an objective being excluded **isn't an aspect to be glad about,** this feature was chosen **during the investigation as it provided extra functionality if the system felt like there wasn't enough to feel complete.** However, as we can see **this was never the case.** Despite this, another reason which lead to the removal of this objective was down to the **PrinterJob** class I would have had to use. When I looked further into the requirements I saw that the whole process surrounds the **Java.Graphics class to draw the item I was going to print,** an aspect I strongly dislike using. After immediately seeing that I would have to solely work with this class **I knew the feature wasn't going to be incorporated**. The last major reason was **timing,** as the aspect was going to be an inclusion to **try and add further functionality if needed**. I knew if I had too much to do and/or too little time left the feature was going to be excluded **regardless as expected both of events became true and as a result the feature was omitted.**

Remember Me

Lastly the final objective to get removed from the system during development was to **remember credentials aspect to the login system**. While the other features were planned from the beginning to be included **this feature was designed during the post prototype phase of the project**. Here the feature would allow users who **frequently access** the system from one device to have the **ability to skip the login system entirely** and allow for **direct access to the system**. This feature, similar to the printing objective was **removed due to the fact that time was getting close to submission and that the feature wasn't necessary to the main aim of the project**. To add to this after the recent inclusion of encryption, **having to deal with file handling this late on was never going to work well in my favour**. As a result the feature was left out due to the **effort needed to get it right was much more than the result I would have got from the final product.**

The Whole System

The last section I wish to discuss will be the **program as a whole**. Overall I feel **the system is more or less great**. It manages to provide a simple but advanced set of features to the user that allow for a **clear solution to the problem at Euxton Hospital**. It does this by performing the document management almost perfectly and focuses on just enough to feel like a complete system but not too much to feel overwhelming. I feel a great general aspect of the system is the file handling and how the data itself is held in the text documents. After finding a solution to the original filing problem during the prototype the approach to deal with customers on a per document basis proved to be a great result for the system. In particular the ability to split many values of data on a single line using specific characters to separate them was a great way of dealing with the inefficient file reader available to me and meant that lists of data could easily be stored together. To add to this another good aspect of the system would be the primary key generation. Coming from the prototype I found that the system in place was perfect. Enough had been done to ensure that every item had a unique identifying key to locate and access it. However, the value was not just a singular key but rather a concentration of many primary keys of the entities parent classes to form compound keys. This meant that the number of actual unique values could be minimised as it was impossible to call an invalid key.

Looking past all the good features there were still plenty of shortcomings the system had also. For instance a main issue I had with the system after finishing development was the fact that all the users from an attribute point of view were practically identical. This poses the question as to whether enough designing went in place to ensure that each user felt distinguished on the system. To add to this the lack of medical fields for the patient also makes me consider if it was even worth bothering with the object hierarchy in the first place. Because of this I will definitely be considering including more fields to the primary users of the system in subsequent versions of the program. Finally the last shortcoming of the system is the colour scheme. Despite aiming to feel simplistic and professional I have instead resulted in a program that feels slightly dull. Despite the highlighting of main boxes I included following the prototype feedback I still think the system lacks that eye catching look and feel I wanted originally. This can be mainly contributed to the lack of colour present. After looking at testing feedback it was clear that other users felt likewise. Not only this but after reinvestigating commercial alternatives I can see that the general similarity is that every program was colourful and lively unlike mine. Again I feel that the solution to this problem would be to introduce a monochromatic colour scheme to re-envisage a better look for the system.

9.32 Potential Improvements

Despite completing the majority of objectives and feeling like a complete system I feel that there are still features that could be introduced to the program regardless to improve its effectiveness. While it goes without saying bugs and glitches should be fixed, nonetheless, here I am suggesting features that would truly enhance the experience of the system and to include past limitations and justify why they might be included. In terms of searching and sorting I feel there is no major improvements that can be made that would affect how the system works. Coming from the test data searches using a linear approach were only a few seconds behind binary searches, the upheaval to replace such a system would be inconsequential for the results that we would see and so keeping the processes the same seems the best approach to go with. While there were countless ideas in my mind when writing this, here are some of the strongest improvements and inclusions I would introduce if I had the chance to improve effectiveness of the system.

Enhancements to Documents

For this feature the document would be greatly improved upon allowing for a full template experience with the current system. First, **documents should be allowed to be edited and deleted** when necessary. This would allow changes to be made if needed and would mean that **errors can be removed**. This was a feature suggested during testing as a response to an error made when the tester created a document and wanted to rectify it. Another major aspect that would be introduced would be the inclusion of **multiple pages** like seen in the document proof provided early on during the investigation. In reality **a single document might not be enough to hold all the information required** and so more pages might be needed. I feel with my new familiarity with the document listeners I think I might be able to include such a complex task. Lastly the main aspect I want to see included would be the **overhauling of the creation aspect**. I feel the **four templates** while a great start **are very rudimentary** and can be improved on vastly as in reality there are obviously not only four documents used at Euxton. What I would like to see happen is that the **ability to create more document types** should be introduced and the ability to **customise the document** and **save different versions** should be a possibility. This became an idea during the development of the prototype but was put aside to focus on more important issues, after looking through old notes the idea has resurfaced.

Major inclusions to the Booking aspect

The main idea to improve the appointment section comes in three parts. First would be to centre the **staff account as the main entity to deal with bookings.** Taking inspiration from Praxis I feel that a drag and drop schedular could really work well and would **allow for better management of bookings** other than a list of appointments that is unable to be edited. In my version I would see the idea as a good way for staff to retain some identity where they can manage patients who have checked in and out and **can allow for the introduction of a receptionist**, a possible new child class to the staff entity. This would allow for **better organisation of bookings and give consultants an improved idea of when an appointment is occurring**. The next idea comes in the form of **more fields being included for bookings**, a key criticism from testing was that it seemed too easy to create an appointment by only having to select a time, date and room for a user. Here I would attempt to provide more details to these sections and include reasons for why the visit is necessary in the first place. This leads nicely to the last section, like the audit log for employees, **visits will now be officially recorded and documented**, using the fields entered by both parties the system will now keep track of appointments that have occurred, the idea was given to me by watching a training video for the program Cloud Care whilst comparing other commercial software this would allow for long term tracking of bookings and would greatly aid new sections I will discuss. However with all these changes to the system I feel that minor file changes isn't enough to allow for such large features like this to prevent data redundancy the system will actually relocate all bookings to an isolated file on the system. By doing so we will be able **prevent loss of integrity and will allow an easier approach with data handling** by not having to deal with updating two files every time.

Interface Overhauls

I feel the major aspect to address, once again, would be management entity. I feel that **the inclusion now of a graphical interface for the management entity** will now be a major priority, while initially thinking that a text based interface would suit the needs of the entity much better, **I now feel that this has come at a cost to presentation,** this can be supported by Alex during the early beta testing who it made clear that **the system looked like it was rushed at the last minute.** As a result, this provided enough reason to carry out the changes as the user experience is a **major factor into initial impressions.** The next major change would be the **reworking of the layout and scaling of components.** Like every other commercial program their software allows for the size of the window to be scalable, because of this **I intend to achieve the same affect.** While initially a limitation due to my lack of knowledge using layout managers, with **BoxLayout** being eventually used during the actual development I think it won't be too much of a task to get components to fit the screen. Finally the last aspect worth mentioning would be the **colour scheme redesign.** Overall the current design is **bland,** my plan for future iterations would be for **a monochromatic appearance,** instead of using a mainly grey background I would instead explore many variations of greys, whites and blacks. I **think the focus would also be to introduce a layering affect.** Not only would this **differentiate the backgrounds** adding variety to every panel but it would also **add colour.** The main goal of this colour change **will be to retain user interest in the look and feel of the system by including a more interesting format I think this will help achieve this.**

Further Attribute Inclusions

For this section in order to make the system more effective overall I think **more attributes ultimately need to make their way onto the system for all entities.** Currently I feel there **isn't enough to distinguish entities other than the actions they perform;** this creates a situation where everyone **feels the same** from a data handling point of view and questions the means I went through to make the **object hierarchy a usable aspect design.** Therefore I feel for the patient, **specific information like BMI** score or **the list of immunisations** should be included to show to the user that **medical information is needed.** Another set of fields that could be included would be the patient's **local clinic and referring doctor,** while originally a **limitation** excluded this, I feel we could now use it since we are looking for attributes to include. This leads on well to the next improvement I want to **include, as a result of the new fields we will need to include new panels and text boxes to allow to the data to be amended.** This was an issue because during testing **when a tester wanted to alter the consultant's wage they were unable to do so as no panel was in place to allow this to occur.** As a result we will need unique panels for each user type to allow these fields to be inspected and altered. The last aspect I aim to discuss is the largest, now that I aim to **incorporate a graphical interface to the management entity** I only now feel it's necessary that a **true account experience is introduced as well.** Here the management entity will now be able to **alter their normal demographic information like any other user.** I want this to happen now due to the fact **I want the user to feel like another main entity of the system rather than a tool to just create accounts on.**

Validation Overhaul

Even more another section I feel that could greatly **improve the effectiveness** of the system would be to **address the validation.** The first section I would change would be to **remove the individual validation methods that all check the same fields just for different entities,** this just **wastes space** on the system and is **unnecessary** as the same **process occurs in at least three different methods.** Instead I will pass the entity's attributes through the **user class and from there will validate the data through a singular method** ensuring that the fields are correct. Not only will this **reduce the number of lines of code** but will **centralise the validation process to one method.** Now from this we can then proceed to work on the validation methods for the particular entities using the new fields **I will include for all the users as suggested in the prior paragraph.** Now that the location of the validation methods have been sorted I want focus on the methods themselves. **While this will take some work to carry out the validation methods for the system are getting replaced completely,** instead of returning immediately after a value is found to be false the rest of the **available fields** are going to be checked and so all of the **erroneous values can be informed to the user** in **one** popup box. This was picked up during testing and instead of being a convenience it was **apparently more of a nuisance as it made it harder to enter data not knowing what other fields were wrong.** Finally to help aid this feature all the text boxes that are incorrect will be changed so that the outline will turn red also indicating an error was found, as a result improving efficiency as the exact fields can be identified without upheaval.

Notification Expansions

The final major aspect I want to discuss here is that **I feel the notifications on my system can use CareCloud as an inspiration** and use the notifications as a **primary source of communication** on the program. Because of this, the first thing I want to do for this feature is **make notifications common throughout all the entities,** this will mean **moving all the methods and attributes to the user class from the patient child class.** This was actually a piece of **feedback I was given during beta testing** as the tester thought that only the patient receiving notifications was **a bad idea if they couldn't do anything with the information,** as a result the changes are happening. The next major aspect will be that **all the users can now create notifications for any other user on the system if relevant,** for instance a consultant can **create a notification for all patients** and fellow **staff** accounts but patients can **only get in contact with the associated consultants** and **staff** account. Similar to the 'Notes' aspect to CareCloud this feature will **allow users to directly communicate with each other.** The final aspect is a minor resolving of a few issues I had with the notifications as a whole: First, the ability to **mass delete notifications will be introduced,** this was an issue I had when **I found it annoying** to delete over 20 notifications on the system, I feel a **clear-all button will be perfect to resolve the issue.** Second, instead of clearing the list from file when the window is closed the system will now **perform all notification actions in real time with the rest of the system.** Despite the extra demand on resources **I feel it will be worth it.**

Quality of Life Additions (improving the user experience)

While not as large as the rest of the other features, these are smaller aspects that would still greatly improve the user experience and so will be included, as some of these I am really keen to implement:

- **Collapsible sections in the contact bar,** like Notepad++, **small minimise buttons could close off irrelevant sections** and reduce the amount space boxes like the address section take up on the contact panel. The idea could also support the use of **a widget like experience,** similar to the feature included **Apple software.** The idea would customise how a user views their account information in the contact bar by ordering the list their information appears being able to add or remove as many fields as they want.
- Leading on from the last section **a settings and preferences panel could be introduced** to house the formatting ideas and aspects of the system, it also keep an array of other features like the next section I will address.
- **Custom passwords,** not much is really needed to describe this but a set of rules for security would be established to ensure standards are met. To add to this **double entry verification would also be needed** to ensure data has been copied correctly from one medium to the another.
- **Date picker enhancements,** here I will look into source code and will make the date picker a permanent fixture to the panel, **without the need to press a button.** This was looked into during development but **seemed that everything revolved around the window that appeared and so it wasn't as simple as just moving items over.**
- The final aspect would be **Date highlighting,** an aspect I always wanted to include but never got chance. This feature will see dates highlighted different colours. For **dates that are full they will be red, nearly full dates to be orange** and **otherwise green to symbolise availability,** with the exception of **unavailable dates being white.** This should help increase visibility to what times are available as patients will be able to judge whether they can go or not.

9.4 Strengths & Weaknesses during Design & Development

System Focuses

Coming into the project I pre-emptively knew from the start what problem the software was going to resolve because of this I made sure that the focus for the system was **perfect from the beginning**. As we were dealing with patient documents the **ultimate aim was** for both the optimal **user experience** and **efficiency** in the tasks it would perform. We can see that the **minimal loading of data from file** being a **prime example** for **effectiveness** of the software where only needed information is read from file **as a result keeping memory use low**. We also see the focus on the user interface **in the design and PPROD documents** which show I was also **fixating on the user experience** by putting a lot of attention on the presentation for the panels and how the layout would appear. However in certain cases the inclusion of both aims didn't work out in my favour at all. A key example of this being the management interface, **this compromise to both aims ultimately backfired and ended up appeasing no-one**. However, as we are dealing with many entities on the system I don't feel this was a worrisome issue, as it was **intentionally planed** that **users who will frequent** the system **often** will have their entity **revolve around efficiency** for instance the **consultant**. As for the users who will **occasionally** make **use of the system** the **user experience** was the focus, for example the patient entity. Despite concentrating on these aspects though I feel that other aims like data handling weren't neglected as the final product shows a well-rounded system. In the end I ensured that the functionality of the system came first regardless of the aims I wanted to focus on in particular.

The Project, Research and Investigations

Overall **the project was very successful** as I think it was a **wise choice** to have gone and addressed the **document management** issue for the program, as a result both me and Euxton Hospital have benefited from this very much. This can be put down to **the problem I set out to fix** and **the insight I had at Euxton** to help me resolve it. What I truly believe to be the main reason the program did so well was because of the **thorough investigating I performed** with **eight** methods of investigation (**MOI**) covering **all four of the different approaches**. Because I took the time to carry the research into the problem, for instance the **three separate investigations into documentation** alongside the **months** I spent there carrying out my job, it meant I definitely knew **enough research had been established** to understand the **problem inside and out.** Because of this I was able to formulate an **appropriate response to what I saw**. The resultant product created clearly provides a valid solution to the **laborious** situation created by the presence of physical files in medical records.

Looking at it now the issue plaguing Euxton seems **perfect** for the project due to its **relevant size**, **uniqueness** and **required skillset to solve it.** The size of the project **felt large enough** were the document management could **act as the centrepiece** to the program and could **incorporate subsidiary** features **if necessary**, however, it felt small enough where I clearly **wasn't disadvantaging myself** trying to carry out an impossible feat. Next was uniqueness, while slightly minor in comparison I still felt **it was a good enough reason to have gone with the project**. While clearly not the only example of an **Electronic Healthcare Record system** (EHR) I know **providing a unique solution was an important aspect of the project**. By having a large enough concept I could **allow for inspiration** from other examples like idea for the **contact bar** in **CareCloud** but feel small enough where **a sense of independence** could be established and not produce a program that has been developed a countless number of other times. Finally we have skillset, for this aspect I wanted to find a problem **I could** really **demonstrate my abilities as a programmer** but **not feel overwhelmed at the requirements** I would need to make that happen. This really reduced my available options, for instance due to the **minimal reliance on external packages** everything would need to be developed **by myself** and so I would need to provide a solution that **took full advantage of my knowledge** of Java and its many associated classes, luckily this was the program to do so.

Project Size and Objectives

I feel the **scope of the system** was **perfect** and included **enough depth to resolve the issues** present at Euxton, to add to this I feel that **none** of the **objectives** felt **shoehorned** in **containing little purpose** as every aspect felt it belonged there. I think this is the case due to the fact that **a strong** enough of an **idea** was available and meant that rather than trying to think of features, I was on the opposite side of the situation and **had too many I had to omit,** hence the lengthy section in 9.32. Having **multiple iterations** and **one to one talks with people** being able to **collaborate** on **suggestions** allowed **for a good idea** of what I had time to include and not include. Overall I feel the scope was strong enough where I could have a clear direction where I was heading but didn't feel that I was constricted to a singular plan and had room to experiment with ideas

Without a **shadow of a doubt** however I included **too many objectives**. I knew for certain only a few days into the design document that **I had given myself too much to do.** The reason for this was due to the fact that **I never wanted the system too thin** where if I finished the system early and it **felt incomplete,** I would be unable to **introduce major features** so **late into development.** My thinking was that I knew if I had too much to handle **I could omit any feature that felt too minor to keep** at the expense of functionality and performance. While overall **working exactly as intended** it was clearly **not the right method to have gone with.** However, this was a **tough call to make due** to my **inexperience** regarding how much **work I could take on** with what I knew at that **moment of time.** I shouldn't be too annoyed with myself as **only three** objectives were eventually removed, if I had **forced** these features into the program the **reliability and robustness** would have seen a **massive decline** as those are a **higher priority** and I can't be too harsh on myself. To add to this I feel the objectives themselves while **fairly large to be considered a single task** and were **perfect, a great example** would be **ID generation,** a task large enough to **subdivided into multiple objectives** but small enough it is a **reasonable task** to keep in a **singular method**. Finally, by having many **attempts to get them right** I think the final version was the set **I was most pleased with** because of them having a **clear set of features** I wanted included. This allowed for a **sense of structure** which I could follow when developing the system and ultimately use a check list **to see progress I was making**.

Timings and deadlines

This was arguably the **hardest aspect of the project** for me despite being mainly **my fault** due to my fai**lings to organise my time efficiently.** I am glad to say that **no work** as of yet has been submitted **late** and **below a level of quality** I strive for. To begin the first year of college working on the project was **controllable** in regard to time management and deadlines, keeping in mind the project **only** accounted for only **20%** of my final grade **I still managed** to make the time for coursework due to the work being **nothing too time consuming.** However as I moved further into the project **the work took more of my efforts** eventually **overcasting** the time spent on **other subjects significantly**. While I just managed during summer, starting early on the prototype I knew **I wouldn't be graced with weeks of free time** when it came to development of the final program and eventually **took advantage** of this. Things ultimately **got worse around Christmas** where we not only had **mocks** but the **final software was due** a couple of weeks later. While trying to work on both wasn't an option I had to do the **right thing** and **postpone** development until **mocks were over.** However by missing out on three weeks of work **it came at a cost** because of this I was **behind schedule** and meant I had to **put in extra work** and focus on the code to **ensure it was perfect.** This essentially meant spending **all night** ensuring the software was at a **presentable stage.** Once the program was finished we then had the next major issue, the **testing document**. While there **wasn't** anything **too difficult to write** the entire process was really **dull** and **painstakingly slow** as I had around 50 objectives in total. I eventually saw even more **nights spent ensuring I could finish the document.** However as I put the effort in I **still managed to submit it before the deadline,** however it certainly came at a cost to both my **mental** and **physical** health and will be certainly **avoid putting myself in a situation like that again.**

Prototyping and Version Control

Overall my **feelings towards the prototype were very hit and miss.** From a conceptual point of view I **think it was amazing** as it clearly **provided a strong overall sense that the idea was solid enough for actual development.** We can see this through the very few changes that were actually introduced from the **post prototype feedback**. The majority of code I used was **eventually discarded** on the final program as it was **unusable for new development.** For example a large section that never made its way onto the new system **was the file management** which had to be completely **redesigned due to how impractical it was to develop with.** This was both a **good and bad aspect.** While other students were able to just carry on using the prototype **I had to work from the ground up again starting mostly from scratch adding more work than what was necessary to ensure a high quality product.** However this also worked in my favour as I could ensure everything was compatible with each section before introducing more features. **Despite the large sections** which were discarded some of the features that were **kept are still highly prominent in the system today** like the **panel navigation aspect** and the **majority of the entity class methods.** Overall by starting on the prototype from late July it **meant that enough time was invested** to ensure that I could take my time with the software and **focus on what was necessary at that moment.**

An important reason to the success I had overall **would be due to the good habits I got into when working on the project.** The most important one would probably be **version control and document management funnily enough.** My approach was rather different than other students as I had a laptop which meant **transporting my work wasn't an issue hence using emails as a backup never crossed my mind,** nevertheless I always **performed full backups to both a physical medium and the college One Drive.** This would occur after making **large advances in the work** I was doing so I never **lost major progress.** This **proved useful** when the document would occasionally crash for whatever reason by being able to recover a recent version it meant that minimal data loss could be achieved. However as I performed **constant full backups** it meant that the **storage would have been consumed rather quickly.** To help minimise this effect a **third generation grandfather, father and son approach** was employed which **worked really well.** Finally the last aspect I used was **version control** when I would **keep a document for long term storage** I would make sure it was **correctly named and located** on my solid state drive, as a result if I ever had to locate the correct version it could be easily done. For instance when a submission is completed the **final version would be stored in a separate area so I could identify the correct version.** This will eventually make **cleaning up documents for the final submission be an easy process** as I know the correct version is being used.

Project Approach and Analysis

Moving on, one aspect of development that saw lots of improvement was my **ability to critique and analyse my own and other's work,** while I still have more I can improve upon **I can still see that progress has been made.** Initially during earlier sections of the project when it came to assessing my own work **I naturally assumed everything was acceptable,** however this was sometimes **not the case** as I may have been **overconfident.** A good example would be the **initial concept and discussion stage** where I had to talk about my ideas with other students, while I thought all my ideas were **useable not everyone else thought likewise.** As we moved further into the project **I started to become aware that not everything I would produce would be of usable quality,** for instance the file management during the prototype. Here I truly had realised that the product, **despite being a prototype,** wasn't usable and that **large changes had to be made.** At this point I had to **look back and critically judge what worked** and what didn't as at that moment as I would have to start from scratch again. Finally towards the end of the project, especially during the writing of this document in particular, **I have definitely had to open up more and analyse my system thoroughly** to reflect on every aspect of the project. To summarise while I think initially I may have **struggled to criticize work I now feel as I have made errors I have managed to understand the mistakes** and as a result use those issues to resolve the problem whilst being able to analyse them critically and reflect on what has been produced.

Another **important aspect to development was the methodology I tried to follow.** As I wasn't creating a **commercial product or collaborating in a group effort** my approach to the work was **nearly unbounded,** excluding the **linear fashion for major sections** like the prototype or design document, I was **open to work on the project in my own way**. Because of this by having the freedom to go about things on my own **I took advantage of it and utilised the Agile development methodology** to work on the project. While never really using the main draw of Agile going back to old documents and making changes. It was **nice to know that I had the ability to make alterations if need be.** I felt a strong advantage to this approach was that **I could plan my ideas for the document beforehand** instead of having to follow the document in a static fashion I could get my ideas down early and then **make changes if they were needed,** for example during the design stage a large amount of **the document needed to be consistent** I specifically remember having to **keep making refinements to the pseudocode as I worked on the panel designs for the system before submission.** However, in some cases I did **deviate from the methodology**. For instance as **I already had an idea in mind it meant that a lot of the appeal to using the methodology was missing** as I never had major modifications to make. However looking at my other choices of Waterfall and the V shaped model, a methodology suggested to me early on, **I feel it was for the best that I went with Agile.**

Learning experience

Overall, excluding testing I found the **entire experience very rewarding and enjoyable**. The project was a **great view into the work I might end up doing if I wanted to study computer science further or pursue a career in the software development industry**. In particular I found the chance to use my understanding of Java to be a **highly gratifying task,** while I could have **managed** my **time significantly better, avoiding having to spend many entire nights getting some of the work done.** I found the **process pleasant enough to remain focused.** Other than coding the other aspect that really appealed to me was **designing the system**. Regardless of **having a second attempt** to get the file reading usable in the end **I enjoyed creating both the front and back end to the program** and would **love to go back to create the document again if I had the chance.** The sections that I really didn't enjoy however was **testing and research into other commercial alternatives.** I **hated testing** due to the **tedious work and the painstaking documentation that took many hours to fill out.** The ordeal left me **never wanting to perform testing again to be honest**. Whilst for research into other EHRs I **didn't particularly enjoy how much research I needed to perform** and the **lack of information** available. What made the task worse was how much the developers wanted me to book a consultation to get full access to review all the features, this led to almost entire sections having to be avoided. Despite the entire project **accumulating to 20% of my grade** I was happy with the end result overall and felt my efforts had paid off.

Regardless if I was to do well or not, I know that **I lacked the observational judgment to make good choices and eventually overloaded myself with too much work,** eventually forcing objectives to be omitted. One instance was when I **grossly overlooked** the work by creating many **too objectives,** as a result a few objectives would have less than **20 lines of code** while others where **over 200,** clearly **showing I hadn't the experience yet to make an informed decision**. Another skill I lacked was **time management,** a **crucial requirement for project this size,** being unable make use of time ultimately led to **situations where I was overwhelmed** with work and had to leave features out for the sake of getting the work to a usable standard, again another unideal flaw to possess. T**he last major skill briefly mentioned before was the ability to give feedback to others,** here after being **too polite** I was unable to give constructive feedback if it meant criticizing other people's current work, while not ruining my work in any way it meant others **were unable to hear ideas they needed** and were **disadvantaged because of me,** unfortunately things haven't improved too much since then.

Overall, **I believe I have gained some new sills from the experience and have made strides in some of my pre-existing ones**. Despite not initially being strong at this, I now **feel confident** to assess my own work and think I could judge **if alterations need to occur,** this comes after months of having to reflect on old work and then make decisions based from them. Another skill I have improved upon was my **knowledge of Java and the ability to use the associated online API,** after having to extensively develop and design the system I believe to have seen major **advancements in the areas**. Without a doubt I have **improved my understanding of the language,** a good example to show this would be the use of the **dynamic declaration of objects,** a feature I would never have been able to implement prior. This comes as a result of me learning how **Java utilises an object orientated paradigm,** from this I was able to **incorporate the aspect with little upheaval**. Even more, the ability to **use the API** to find what I needed was a great aspect to learn how to use, this has **meant less reliance** on other documents such as the **introductory booklet we were given early on in the course.** Lastly the final skill I gained was the **dealing with deadlines,** while I struggled to manage my time **towards the end** of the project it never **resulted in a late submission,** this skill will come in handy if I ever need to submit work, which is **highly likely.** Regardless the largest improvement I have gained would clearly be **experience by having the chance to carry out a project like this it has meant I have managed to get an insight into the software development cycle,** for instance getting to experience **common practises like time crunch and many deadlines.** Despite how unrealistic certain aspects may have been I feel the **overall process has given me great practise** for what awaits me if I was to pursue computer science in further education.

9.5 Future Changes of Approach

Time management and constrictions

For future projects one aspect I would like to see different would be **how my time is spent.** First I would **introduce limitations to how much time would be spent on a single task**. By creating a deadline to get tasks done it **would allow for progress to be continuously made** and would put **pressure on myself to get the work done.** It would also mean that my efforts could be **equally distributed,** something that **never happened** during development. For instance during changeover from the prototype to **the final program it took 4 weeks instead of 2,** clearly time which could have been spent **being more productive.** Moving on, another aspect I would carry out would be to allow **more time to test,** by giving myself **a few more days, possibly a week,** I feel I would be have been **less stressed** to ensure the program was **up to standards for submission.** It would mean that **less bugs would be present** at the end as a more **thorough testing process can be carried out**. The next inclusion I would make would be to **give myself breaks;** I know for certain this **hardly happened during the actual development.** This would greatly reduce the chance of experiencing burnout, even more its **not healthy to be confined constantly to a des**k so it would be in my best interest to **designate time off occasionally to relax**. Finally I would try and start more work earlier. Here I would attempt to **get ahead on work if I was to get the chance.** For instance when I should have started on testing I was still applying the finishing aspects to the program ultimately leaving me with more work to do in less time. If I was to move onto testing it would have seen less trouble to ensure the document had been finished.

Workload

Now I will address my **biggest weakness of the entire project, giving myself too much work.** If I was able to do things differently I **would definitely ensure that I wouldn't give myself too many objectives** to work on. What happened in my project was that **instead of trying to do a reasonable amount of work I overcompensated by introducing many minor objectives**. As a result I included **too many** where I was in a situation that some had to **be omitted to** ensure I had time to **properly test.** I think with my experience now my **capabilities have been clearly realised** and so setting myself work shouldn't be as hard as a task to carry out. The next change I will include is that I would **ensure that all my objectives were of a similar size.** The issue with mine originally was that **some felt so minor they became insignificant in comparison to most of the others despite requiring the same amount of documentation,** this meant they felt unnecessary as they played a less important role in the grander scheme of things, regardless this will aim to reduce the feeling.

Prototype

The next aspect I would change would be the **prototype.** First when deciding on **what is to be included,** I would make sure that every **aspect involved is essential and necessary.** Here I would only see major **objectives that would demonstrate my concept to be included in the prototype**. The reset would be included at a **later point when we have time to.** The original issue I had was that I tried to include as many objectives as I could and as a result of this **nothing was usable for the final system due to how unreliable my code was,** this eventually put me in a situation were starting from the **beginning was the best approach forward** and left me with **lots of time wasted,** hence the desire to **limit what is included in the prototype.** To add to this the next change would be also to see the **focus on experimenting with ideas instead of trying to develop the final product** at this stage, as we are developing a prototype we can allow for different ideas to be explored, if not obvious yet, **this never happened with my project** and led to me using the **initial attempts** for every objective I had. If I was to test my ideas it would allow for the best to be included rather than choosing the first one that hardly works. For instance **during development I originally struggled** to include aspects like the mouse and event listeners as I was focused on getting as much into the prototype. Overall it meant the **opportunity to understand the features was missed** and as a result had to take up actual **development time to learn the basics** for simple aspects.

Development

With my approach to development one aspect I would like to change would be my **handling of searches**. Next time I would make sure from the **beginning that all search algorithms would incorporate the ability to search all fields on the system rather than just a singular one**. This is the case due to an issue **I had brought up in section 9.4** where I **struggled to perfect most objectives** early on in the prototype which as a result meant that halfway through the actual development **many hours had to be spent ensuring that the search was actually usable,** by getting it correct from the start this wouldn't happen. The next issue I would like to address would be that I want to take **advantage of the Agile methodology more,** in particular **I would want myself to go back to old documents and have the chance to make changes when necessary** rather than sticking with the mistake to avoid the possibility of making continuity errors in my documentation. This was a bi**g issue with my projec**t and could have possibly **allowed for the redesign of the files** to be rectified much sooner **rather than during refinements of the prototype**. Moving on, another change I would like to make would be my **development of entities,** a major issue I experienced was that **I developed the user entities far too late** and so when I needed to include further attributes it meant that I **had to go through possibly 40 different files to include the new field**s to avoid null pointer errors. By working on it **early with minimal testing data** it would ensure that **changes could be made swiftly and effortlessly**. This leads on nicely to my last point for development, an error that occurred in **both the prototype and actual development** was that I kept **interweaving both the front and backends of the program**. This has become such a **large problem** that it will be **hard to debug certain aspects or introduce large changes** and will result in **me having to work from scratch again**. Specifically the change I would like to see is that **very little is shared between the Gui class** and the backend classes **minimising the reliance** each has on the other. In particular **I would like all entity processes** to be located all in the entity files. This should **reduce the presence** of entity processes and attributes in the Gui class which would resolve many of the issues mentioned.

Feedback

Finally, **the last major section is feedback,** the first change I would introduce would be **more reliance on verbal and written feedback rather than quantitative data**. While it does show a **clear indication** to what the **general consensus** is, it **doesn't actually help me** in producing ways to improve but rather just **points out the flaws.** This clearly doesn't help **unless advice to rectify them is given**. This was a problem during **development** as while I had many graphs showing a clear correlation between my system and the tester opinion it was **slightly redundant due to the fact I had no ideas suggested to improve on them.** To add to this another **change I would make is that I would want to focus on more beta testing,** despite what was actually done, **I felt this wasn't enough**. By including more people to give opinions it **would mean a less unbiased opinion would be given** due to the fact **many testers for the final system also reviewed the prototype** and so had pre**-existing feelings** towards it. Finally the last change I would make would be that I would need to become **more critical of not just my work but others.** While slightly off topic, I feel it is important that I realise I am helping **no-one if I were to give bad feedback** if I just say that no changes needed to be made. This links back to myself as it will also help me spot flaws in my work.