Software Engineering 1 Assignment

C14337041

Introduction: This report aims to cover a solution to a library software system. The report will include C.A.S.E model(s) that have been modelled in Argo UML. The report will look at a simple design for borrowing and returning books to and from the library itself. All diagrams within the model will come with an explanation so that the user can gain a better understanding of the inner working of the library software, also so the user will be able to visually comprehend the fundamentals behind the software.

Requirements:

1. An understanding of a library system.
2. Basic knowledge of C.A.S.E models
3. The ability to code in Java / read and interpret Java code.
4. Must know how systems and user operate together.

Library Systems: All library systems are similar, if not identical to one and other. They all do the same things but for different people. Some examples of what a system may do are as follows:

- Allow a user to borrow a book.
- Allow a user to return a book.
- Allow a user to reserve a book.
- Remind a user when a book is due.
- Be able to track all books that are currently reserved / borrowed.
- Keep track of how many copies of a book a library has.
- Tell staff when a user has returned a book past a due date.

On their own, each one of these tasks seems quite accomplishable. If you try to take on more than one task at a time however, it can become complicated for people. This is why it is important to factor this in when designing the software for the user. It must be simple to use an also procedural so that the user doesn't get overwhelmed when trying to accomplish a simple task.

C.A.S.E Models: These are a great way of taking something complicated such as an application / software design and making them simple, clear cut and easy to understand. They make the developer's job ten times easier by allowing them to see clearly what has to be done. It is also a great way to take the technical side out of things for clients. It allows them to look at exactly what their system will do graphically without having to listen to any technical jargon that they might get lost in. An example of diagrams used within this case model is as follows:

- Activity diagrams
- Use case diagrams
- Class diagrams
- Sequence diagrams
- State charts

All of the above diagrams offer different ways to see similar information, but where one diagram might have missed something important, another may have it included and that's why it is important to have multiple diagrams included in this software design report.

Java:

Java itself is a very powerful language. It incorporates things such as inheritance, encapsulation and classes which most Object orientated languages have nowadays. Provided with this report will be a basic piece of Java code to demonstrate the basic workings of the class diagram that will be listed further down.

It is important to use Object orientated languages such as Java for projects like this as the client / user could be dealing with multiple objects at any one time (Books and users to name a few in this case) and the need to accommodate this as simply and efficiently possible is important.

Multiple classes can be used to perform different tasks such as borrowing and returning books, creating new users and updating user info and also tracking and updating book information and it is important to factor this into the design.

Human / System Interaction:

People today want systems designed as simply and efficiently as possible now a days. This is the real challenge for the designers and developers. One the one hand, it is important for the system / software to be excellent functionality wise, but on the other hand, it is also important that the end user is able to benefit from this functionality by being able to use the software / application.

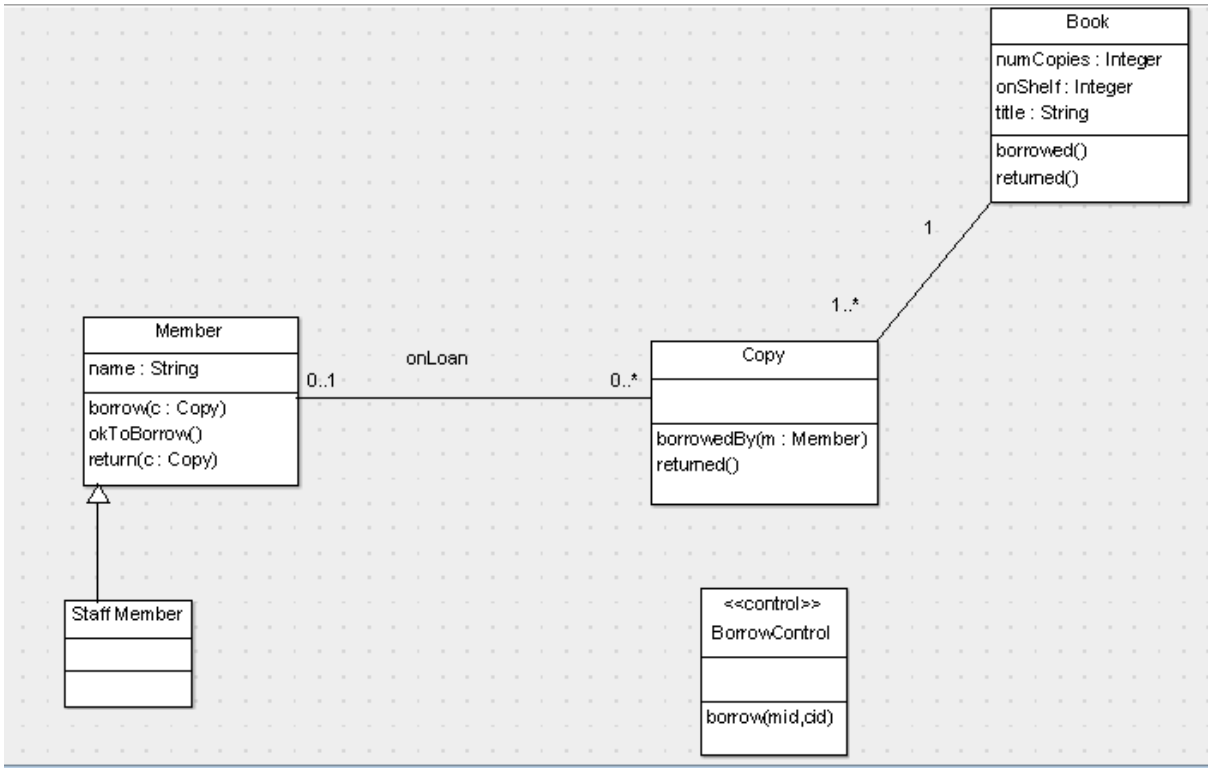
It is important to build the software / system to the client's specification, taking into account all functionality they require. It is also important to incorporate every instance / occurrence of a problem the user may have operating the system / software and try design a simple solution (s) to the problem (s) long before it occurs to maximise the efficiency of the software / system in question. This should be taken into account when designing the software initially to try and stamp out as many problems as possible.

Argo UML C.A.S.E models

Model 1: Borrowing and returning

The following model consists of multiple diagrams that hope to address many problems a developer may have when designing the software by providing a graphic solution to the problem.

Class Diagram:



The class diagram is the heart and brains of the model. It shows the user exactly what needs to be created and how it needs to be laid out. From the above diagram we can see that there are five classes, four of which are linked. The book class is used to store all information about a specific book. Examples of these include Number of copies in the library, the book name and the number of copies on the shelf at any one time.

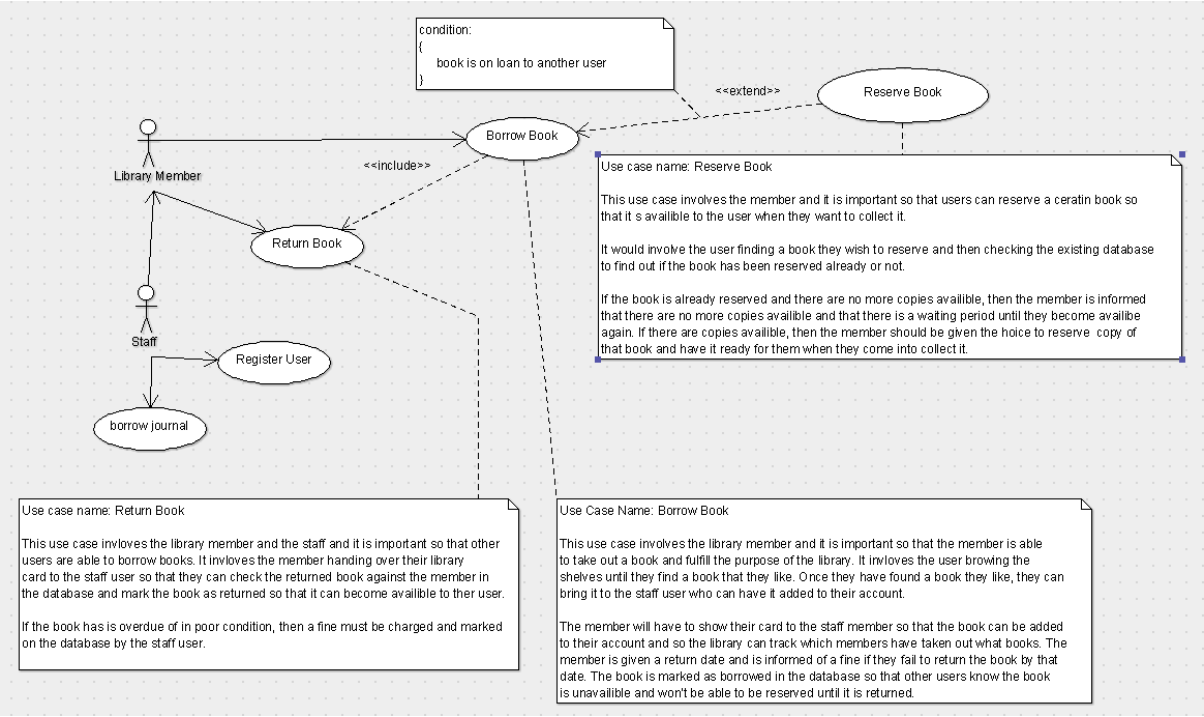
The copy class serves as an identifier which will be associated with the member class. The copy class tells us who has reserved what book and will also allow us to run a return method once the user has returned the book.

The member class stores all the details about individual members, such as their name and whether they are ok to borrow a book or not. Two methods borrow and return can be executed from inside this class when a user chooses to either borrow or return a book. A third method called okToBorrow can be executed to check if the book is ok to be borrowed by the user.

The staff member class is blank for now, however it could feature staff login details that could be used to authenticate the system.

Finally the borrow control class is an independent class that is used to check the availability of a book using a copy id and a member Id, it could return a yes no answer in the form of a 1 or 0 to inform the user of the result.

Use case diagram:

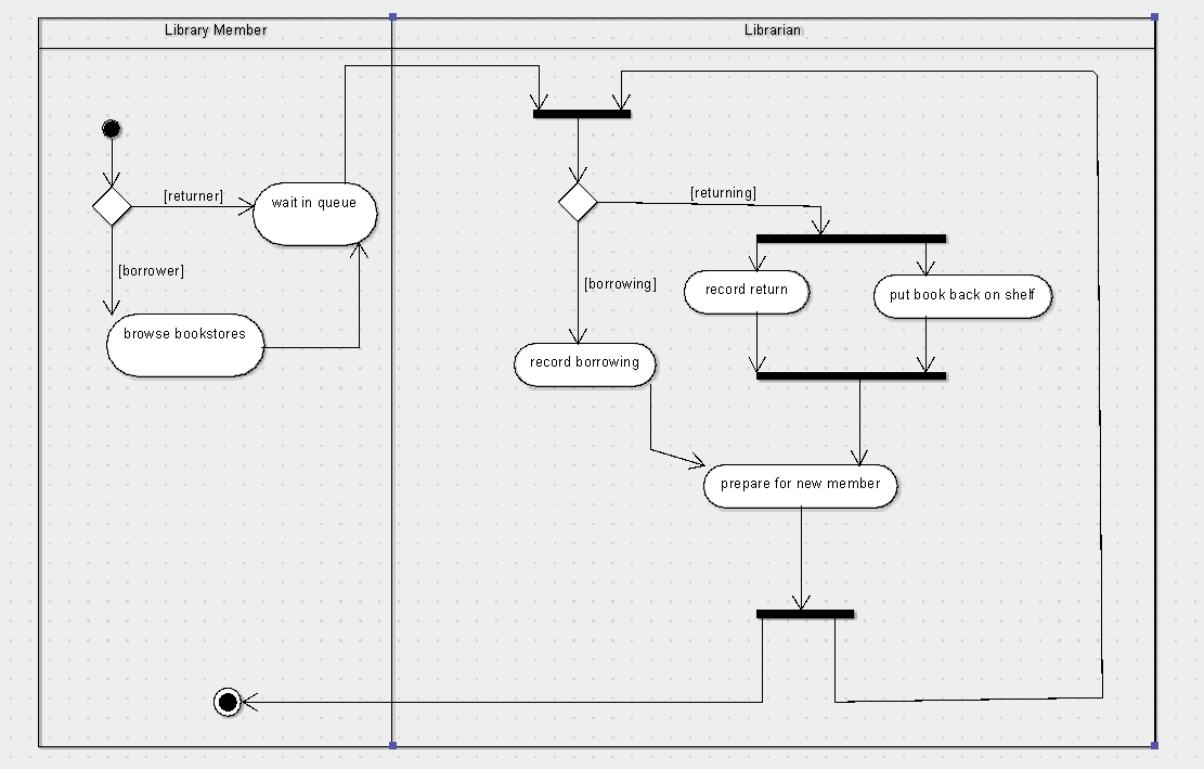


The use case diagram allows the user to see what their role in the system / software will be. From the above simple use case diagram we can see that the two main roles in the diagram are the library member and the library staff member.

The 3 main use cases are The “Return Book”, “Borrow Book” and “Reserve Book” case. Each one is important in their own respect as without one, the other could not function. These main use cases are outlined in more detail in the image above and also in the UML files provided.

The main thing to take from this is that the staff member is key to all of these cases, as without the staff member, they would be no one to process a returned book, borrowed book or reserved book and there would also be no interaction with the user.

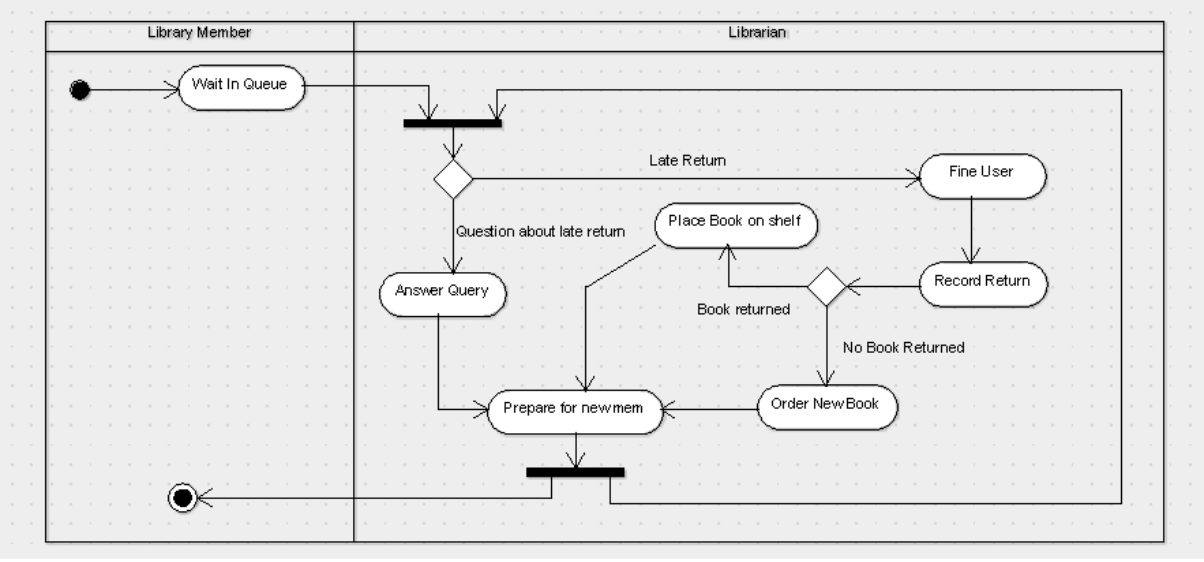
Activity Diagram 1:



Activity diagrams are a great way to map out an event / activity as a graphical representation to help you solve problems step by step and come up with a logical design approach.

The above activity diagram maps the actions taken by both a member and a librarian in the simple case of a user returning and or borrowing a book. There are many options within the activity diagram and it was important that you didn't leave an activity out so that the accuracy of the diagram didn't increase. We can see from the diagram that the member has very little to do, whereas the librarian has a lot on their plate and this is important to take into account in the design processing of the software.

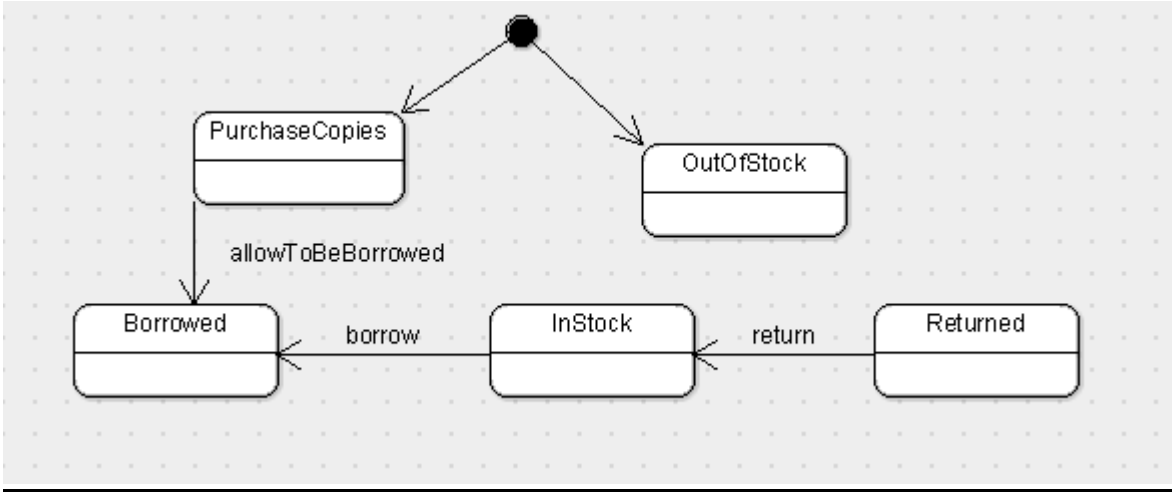
Activity Diagram 2:



The above activity diagram details the actions involved in a member returning a book late or returning a book late but not actually returning the book at all. There are a few decision's made in this diagram so that shows us that the user of the software is going to have to have multiple choices to choose from in the software so that they are able to adapt to the current situation they are in.

Again we can also see that the librarian has a lot to do from the diagram and the member again has very little to do, so this is further indication that we must take this into account when designing the software.

State Diagram 1:

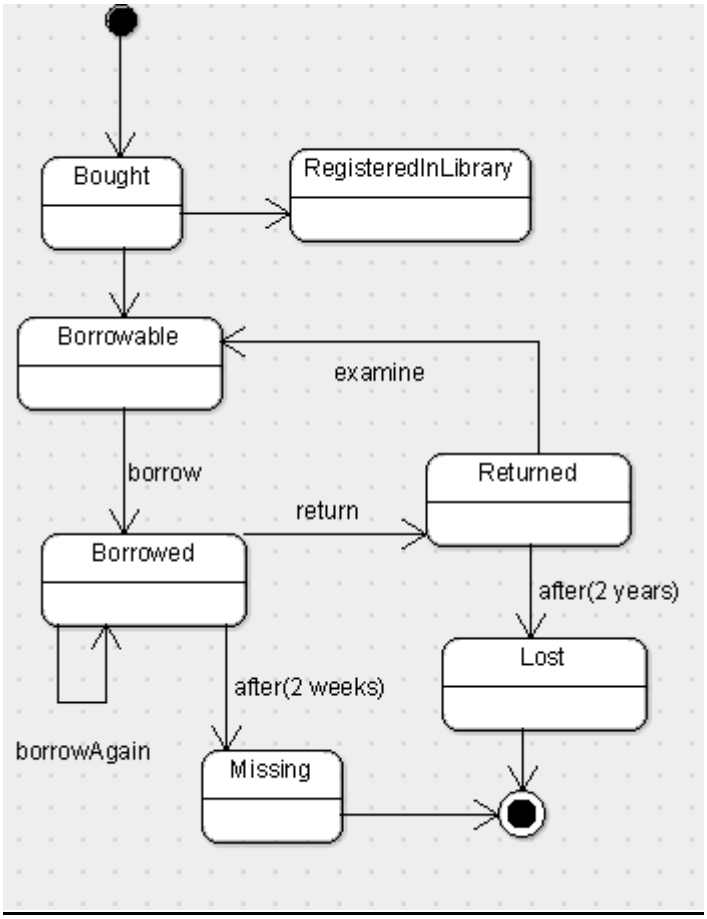


State chart diagrams are important in the software / system design process for a few different reasons. The main reason being that they show the different states a component or object in a piece of software takes on. From this we can see exactly what methods, classes or operations we need to assign to the component / object in order to make these changes happen.

The above diagram shows the different states of a book in the library. If we start and the book is out of stock, we need to go back to the start and order more stock so that we can allow people to borrow them. After which the books can go on to be borrowed. If a book has just been returned, we can mark it as in stock and proceed to allowing in to be borrowed again.

From this diagram it is clear that a database of some kind will have to be implemented in order to keep track of books we have in stock.

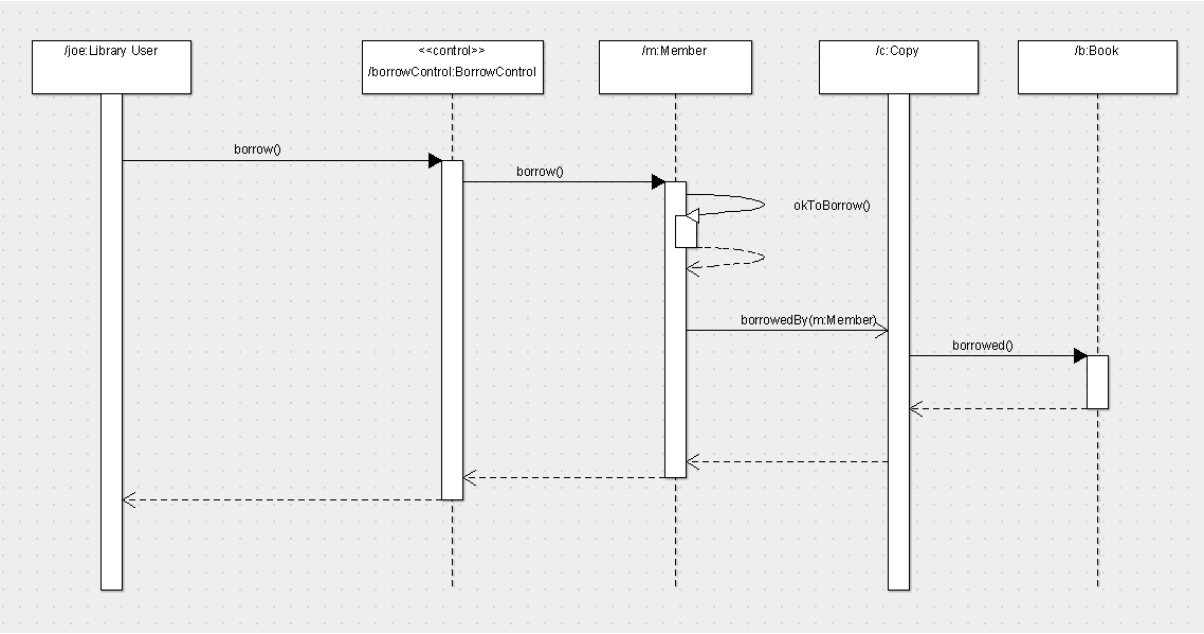
State Diagram 2:



The above State chart diagram shows the different states a copy of a book can go through in the library. We start off with a copy of a book being purchased for the library, have it registered in the library and then marked as borrowable. From there the book can go on to be borrowed by a member and they can re borrow it as long as they like until they return the book where it is examined and marked as borrowable again. If the book hasn't been returned after 2 weeks of its due date then we consider the book Missing and the member who borrowed the book must be contacted and if we have to book returned and we misplace it and after 2 year the book is still missing, then we consider it lost.

From this diagram we can tell that we will need some sort of tracking system on the books so that the library can tell what condition the book is in, where it is and also who has it.

Sequence Diagram 1:



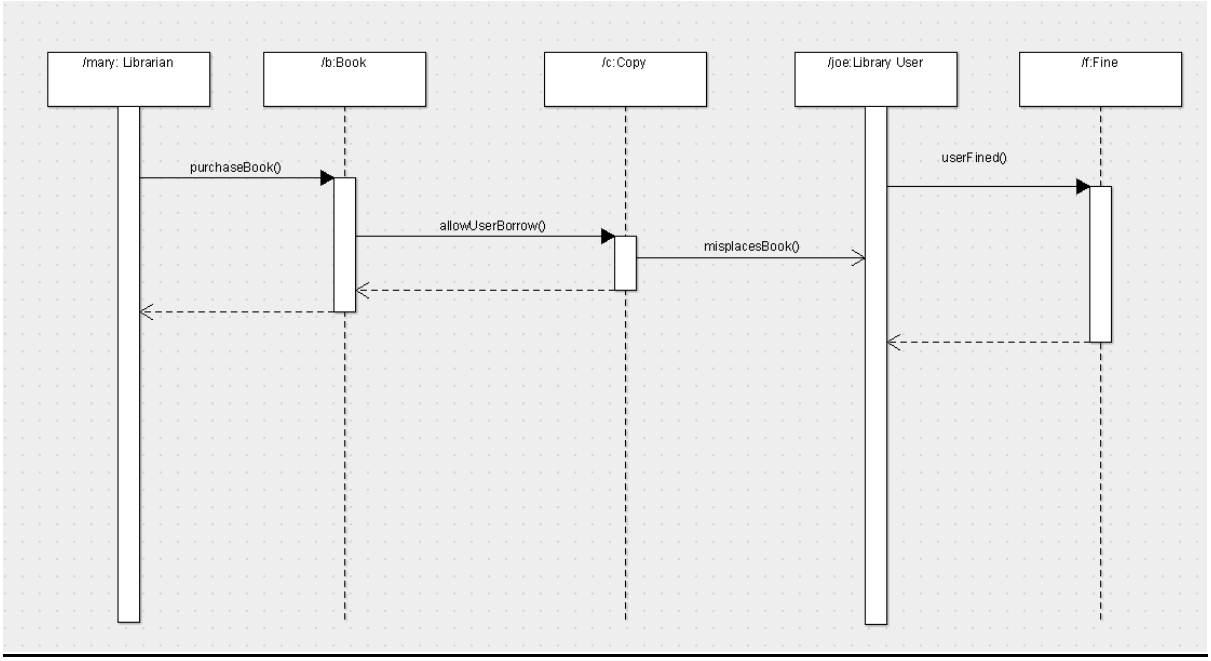
Sequence diagrams are really powerful as they show us how different process or classes operate with one and other within the software / system.

From the above diagram we can see what is involved in taking out a copy of a book. The user interacts with the borrow control through a borrow method. The borrow control interacts with a member by the same method and then itself by using the ok to borrow method. If all of these have successfully interacted with one and other, we move on to the next stage.

The member then interacts with a copy of a book using the borrowed by method and then copy of the book interacts with the book using the borrowed method. Then all the relevant information is passed all the way back to the first process that mad then call.

It is clear that processes will have to link with one and other heavily in this system, so classes will be very important and so too will the concept of inheritance.

Sequence Diagram 2:



The above sequence diagram shows us what's involved in fining a user for not returning a book.

The librarian interacts with the Book using the purchase book method and the book interacts with the copy of that book using the allow user borrow method. Once the copy has been purchased and filed the next step occurs.

The copy of the book interacts with the user Joe using the misplaces book method and then finally Jo interacts with the fine using the User fined method. Then all the relevant info is passed back to Joe so that he can be fined accordingly.

Again from this diagram we can see the importance of inheritance and processes working together, so this will have to be taken into account in the design stage.

Conclusion:

From writing this report I have learned about problems that are faced when creating and designing software and more specifically in this case for a library. There are many problems that must be tackled head of such as what happens if they are multiple branches from a problem and how do we solve them all. I have also learned the importance of C.A.S.E Models and how powerful they are for designing software / systems. Finally I have also learned that it is important to take every aspect into account when designing a system / software such as users and existing systems and how they can interact with one and other as this can make the design process a lot more efficient by taking into account a lot of different outcomes.