Subject: Assignment 2 Guidance

Message:

The architecture required has to support the whole system - this is represented by the 'CCRD use case' of supporting the 'storeLoad' functionality of the system as described in the case study. In terms of satisfying a users intent, 'storeLoad' is the appropriate unit - a user submits a job with the intent of getting their load stored, and at the end of that process the load is stored.

However, somewhat confusingly, I have split that 'storeload' functionality into 2 use cases - 'submit Job', and 'store load'. I think I have explained that I did this because I don't like use cases that 'hang around' - I prefer them to be 'atomic' - i.e. to start, execute, and complete. In terms of how the case study described the system, there is a clear gap between the process of submitting a job, and the process of executing that job and so I chose to split the original 'storeLoad' use case into 2 as described in the 'Asg1 Solutions' and 'Use Case Description' documents.

The point is that the architecture is for the whole system - it has to support the whole storeLoad process - and in terms of how I broke things down, that means it has to support both the front end of the process ('submit job') and the back end of the process (store load - i.e. drive a crane around).

So from the point of view of the architecture (Part A), I want an architecture that will support both 'submit job' and 'store load' as described in the accompanying documents.

However, for Part B, detailed design, I want you to do 'manual store load' and produce detailed (pattern based?) designs for that. The thing is, in the detailed design, the design should be ready for implementation i.e. a programmer should be able to take your design, implement the classes as specified, and the design should support that use case properly - that means you have to deal with user interface **and** persistence issues (and any other Non Functional Reqs(NFR)). So don't forget things like logging and storing information in databases if that's what you decide you need to do.

For Part C, the user interface I want you to consider is the Crane Driver's interface required to support 'manual store load'.

For Part D, we need persistence mechanisms for storing Job, Load, and Storage Slot information, so that the SCS can track job status, knows what is in a load (if it is product aware), and knows where it has put things once they are stored.

Anyway, I can sum this up like so:

1. We want a high level architecture for the whole system that clearly addresses the NFRs.
2. We want a detailed design that demonstrates how the architecture supports the most important use case for the system (in this case the 'manual store load' use case as specified in the Use Case Descriptions document)
3. We want a UI design that clearly demonstrates how the interface will meet the **functional requirements** of the 'manual store load' use case for the crane driver.
4. We want a persistence layer design that shows how the system is going to store it data.

Its quite a big assignment - and there is an extension so that the **Assignment 2 is due on the 11th October** rather than the 4th. However, that is a hard deadline, it won't be possible to return assignments before the exam for anything received after that date.

Just remember - this is a 'signpost' subject - it is more important that you have some idea of the concepts and how to apply them, than it is to get things 'right'.

In this case, the most important things are to remember that an architecture addresses the NFRs. It is important that you can point to some feature of the architecture and say 'this is how we are addressing NFR X'.

Then remember the distinctions between 'functional' and 'technical' components. Functional components group related functionality and arise from the problem domain. Technical components provide the system services that the functional components require in order to meet the NFRs. Roughly speaking functional components group related aspects of end user functionality, while technical components address the NFRs.

What NFR's are involved in delivering the functional aspects of the system? That tells you how the functional components and technical components are connected.

Remember that there are 2 major sets of forces driving an architecture - one is the functional and technical divisions, and the other is the physical topology of the system - how it is going to be deployed. Components have to respect hardware boundaries.

In terms of detailed design its important that it is consistent with the architecture, and that it demonstrates you learned something in terms of how objects interact back in use case realization. I would also really like to see some awareness of software patterns displayed. Coherent and sensible is what I'm really looking for - it doesn't really matter if you don't address every last little alternate flow. Clear, coherent, workable - not comprehensive, complicated, and confused.

In the UI, once again consistency is key. It has to support the use case as described - and it should be consistent with the detailed design. The other important thing is that it should address the functional requirements - so pay particular attention to what information has to be displayed and when. Then make sure it supports the right user inputs, also at the right time. Look and feel are not important, I am most concerned with that it supports the information display and input requirements of the stated use case.

Finally, as far as DAO go, I'm really just looking for an indication that you are aware of what a DAO is, and where it fits in the application. I don't really need a detailed spec of DAO for the whole system. Maybe just Jobs and Loads.

**Important Note:** it is probably impossible to do the detailed design **without** considering the user interface and the persistence layer - you will have to **iterate** through parts B, C, and D to get them consistent. This is the 'following step serves as verification for previous step' theme coming through again.

The Marking Criteria:

 The 3 Parts A, B and C would be marked out of 100. Part D is optional, if you attempt it, the marks you get for it are bonus on top of the marks you achieve. This would boost your total marks. The new marking scheme would look like this: Part A: 35 marks (a:15, b:10, c:10) Part B: 35 marks ( a:12, b:12, c:11) Part C: 30 marks (a:8, b:8, c:8, d:6) Part D: (optional) 20 marks ( a:5, b:5, c:10)