Try to compile/use the source code using the instructions provided. Can you get it up and running? Is anything problematic? Are there steps missing or assumptions made?

No, No huge bugs where found, one index problem easily fixed.

Does the implementation and diagrams conform (do they show the same thing)? Are there any missing relations? Relations in the wrong direction? Wrong relations? Correct UML notation?

The sequencediagram reflects the applications functions in a good way. You can clearly see how the application functions.

Is the Architecture ok?

- Is there a model view separation? – Yes!
- Is the model coupled to the user interface? – No the Model isn't dependant on the user interface.

- Is the model specialized for a certain kind of IU (for example returning formated strings to be printed) – No it can be used on many different applications.

- Are there domain rules in the UI? – No, there is no real logic in the view. It's handled in the model.

Is the requirement of a unique member id correctly done?
Yes, GUID was used.
What is the quality of the implementation/source code?

- Code Standards – The implementation where really good we would rate it an 8/10. We only found one bug which was an index bug when creating a boat.

- Naming – Overall good naming, some names are…Questionable but nothing really worth noting.

- Duplication – No code redundancy was found.
- Dead Code – A few lines of dead code was found,Console.cs lines 13-17 aswell as MemberDAL.cs lines 40-47, the catch is never used.
  Other than that, no dead code found.
- ...

What is the quality of the design? Is it Object Oriented?

- Objects are connected using associations and not with keys/ids. –Yes.
- Is GRASP used correctly? – From what we can see GRASP is used correctly.

- Classes have high cohesion and are not too large or have too much responsibility.
  Yes they have good cohesion, they are split evelnly so no class takes too much responsibility.

- Classes have low coupling and are not too connected to other entities.
  They are not connected in too much of a way, We feel like you could take and apply some of the parts of the code in other systems without more or less any problems
- Avoid the use of static variables or operations as well as global variables.
  None found.
- Avoid hidden dependencies.
  No hidden dependencies found.
- Information should be encapsulated.
  The information sent between classes are encapsulated and should be at no risk of being tweaked with during transfer of data.

- Inspired from the Domain Model.
  We talked to the group and they gave us the Model, after looking at it we can see a resemblance between the Model and the finished application.

- Primitive data types that should really be classes (painted types)
  Not from what we can see while looking at the code.
- ...
  Nothing other to add.

As a developer would the diagrams help you and why/why not?
After that we had found some dead code, we realized that one of the class diagram relations was there only because of this piece of dead code. This relation was in practice never used.

Other than that the diagram is really useful, it provides a good overview of how the program works.

What are the strong points of the design/implementation, what do you think is really good and why?
The code was very effective at it's job. No redundancy and more or less no dead code. It's a really good application and you can see that the design/implementation was really good overall.

What are the weaknesses of the design/implementation, what do you think should be changed and why?

The application should probably have been overlooked one or two more times before finally sending it in for a peer review, there are a few questionable names and some dead code left over since earlier iterations. Other than that nothing is really weak in the application.

Do you think the design/implementation has passed the grade 2 criteria?

Yes without problems.