Instruction

Look at the models, implementation and any accompanying documentation. Try to have an open mind and focus on trying to understand the materials as it is presented.

Test the runnable version of the application in a realistic way. Note any problems/bugs.

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?

The application started fine, while running the application we had some problems. There are a few assumptions made, one example is that you know all the members personalnumbers since this is needed to register a boat. This is not good at all. Also we can't find specific member- is this missing?

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

The naming isn't correct, the program and diagram shows different names for the functions. Other than that nothing really to add on the diagrams.

Is the Architecture ok?

- Is there a model view separation? The view creates objects of user, this breaks the MVC model.
- Is the model coupled to the user interface? No, it goes via the controller.

- Is the model specialized for a certain kind of UI (for example returning formated strings to be printed) – Yes this model can only be used with this UI from the way that it's coded.

- Are there domain rules in the UI? – Yes sadly you validate data in the view.

Is the requirement of a unique member id correctly done?
No, you use Personalnumber as a unique ID, this is not allowed and should have been a separate value
What is the quality of the implementation/source code?

- Code Standards – Pretty bad sadly. Not a lot of comments and we had some troubles understanding the reasoning behind some of the functions. No real "flow".
- Naming – Decent names, there are a few questionable ones but overall they look okay.
- Duplication – Lots of redundancy this is not so good.
- Dead Code - A few empty if(), you create things at some places and never use them later. 186-193-Startview.cs
- ...
  You use Bool quite often, try catch would have been better.

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

- Objects are connected using associations and not with keys/ids. You use keys ID, this is not good and should not be used.

- Is GRASP used correctly? – No you create and validate a lot of data in view that should have been handled in the controller.

- Classes have high cohesion and are not too large or have too much responsibility. Lots of things that should have been in controller is in the view. This is not good.

- Classes have low coupling and are not too connected to other entities. The classes doesn't seem to dependant on eachother.

- Avoid the use of static variables or operations as well as global variables. No global variables found, good!
- Avoid hidden dependencies. Lots of hidden dependencies found, create one object at the start and reference it, don't create one instance in each function.
- Information should be encapsulated. Information is encapsulated. good
- Inspired from the Domain Model. No Domain Model found.
- Primitive data types that should really be classes (painted types) Nothing negative found.
- ... Nothing more to add.

As a developer would the diagrams help you and why/why not?

As we mention before, the names in the diagrams and the names in the program did not match. This made it very confusing to try and find the functions in the program since they had different names.

What are the strong points of the design/implementation, what do you think is really good and why?

We found it good that the person used a real database.

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

The MVC structure was not followed aswell as GRASP. The person breaks the DRY principle a lot.

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

We don't think this assignment well pass the grade 2 at the moment. The code needs a lot of refactoring