Peer Review Workshop 2 – Markus Palm

Problems/bugs

- **Serious issues**
  - Requirement 4 - Unable to edit a member.
  - Requirement 7 - Unable to remove boats from a member if the member has more than 1 boat.
  - Requirement 9 - No persistent data storage.
  - Requirement 10 – No strict Model/View-separation. E.g. model-classes use console.writeline.
- **Other issues**
  - An empty string validates as a name for a new member.
  - When trying to remove a boat from a member with 2 or more boats the program throws an exception.

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?

Result:

- Easy to get the program running, just download and start the exe-file.

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

Result:

- The class diagram looks incomplete and doesn't look like a UML class diagram. (Classes are missing or incorporated into other classes).
- The sequence diagrams are ok. Although they are overly detailed in some parts, and not detailed enough in more important parts.
- The sequence diagrams do not cover all requirements, 6 out of 9 are missing.

Is the requirement of a unique member id correctly done?

Answer: Yes.

What is the quality of the implementation/source code?

- **Code Standards**
  - Code structure varies a bit but it's readable.
- **Naming**
  - Naming is done in a good way.
- **Duplication**
  - We could not find any duplicated code.
- **Dead Code**
  - A couple of segments of commented code was found.

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

- **Objects are connected using associations and not with keys/ids.**
  - Yes.
- **Classes have high cohesion and are not too large or have too much responsibility.**
  - Some classes have too many responsibilities, implementing a Controller would definitely help.
- **Classes have low coupling and are not too connected to other entities.**
  - The level of couplings seems to be good.
- **Avoids the use of static variables or operations as well as global variables.**
  - No static or global variables were found.
- **Avoids hidden dependencies.**
  - No hidden dependencies were found.
- **Information should be encapsulated.**
  - The use of getters and setters encapsulates the information in a good way.
- **Inspired from the Domain Model.**
  - Not sure.
- **Primitive data types that should really be classes.**
  - No.

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

Answer: The sequence diagrams were helpful. The class diagram was confusing and messy.

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

Answer: The finished parts of the executable application works as intended. It was easy to start and run. The source code was easy to navigate due to proper sequence diagrams and good naming.

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

Answer: The application is not finished, does not meet the requirements and does not implement a working MV structure. A more user-friendly interface would have been nice. Proper Model/view separation would also help the implementation a lot.

Do you think the design/implementation has passed the grade 3 criteria?

Answer: No. The application is not finished, does not meet the requirements and does not implement a working MV structure.