# Peer Review Workshop 2

For the domain model made by Erik Hamrin (eh222ve), Julia Sivarsson (jsigc09) and Micael Persson (mp222si)

Daniel Vedin (dv222bk)
2015-10-08

**Application**

Application works well. It did not crash and did what it was supposed to do. Error handling leaves some things to be desired however. For example, the application does not check if a social security number is only numbers. I could enter anything in this field, as long as it was 10 characters long. I for example created a member with 3 spaces as a name and 10 spaces as social security number. Later when this member is listed, all spaces are removed (they still exist in the database however). Entering only letters for the length of a boat results in a length of 0. Neither of these are a requirement for the workshop however but worth noting ☺

**Class diagram**

view.HTMLView seems to be missing from the class diagram.

I feel that the connection between controller.Program and view.NavigationView is not a dependency but an association since controller.Program saves an instance of view.NavigationView. [1, p252]

controller.Program should have a dependency arrow to model.Boat since it uses boat objects in the Editboat() method.

view.Boat is associated with model.dal.BoatRepository, which is not shown in the diagram. It also has a dependency towards model.Boat which is not shown.

view.Member is associated with model.dal.MemberRepository, which is not shown in the diagram. It also has a dependency towards model.Member, but this is shown as an association in the diagram.

model.Member has a listed association with model.Boat, but in reality it is not associated with (or have a dependency towards) model.Boat.

model.Boat has a listed dependency towards model.Member, but in reality it is not dependent (or associated with) model.Member. One might be excused for thinking so, but storing a non-existence "ownerID" in model.Boat does not break the class. The application itself might not be happy about it, but that is another matter entirely.

model.dal.BoatRepository has a dependency towards model.Boat, which is not shown in the diagram.

model.dal.DatabaseConnection is an abstract class but is not marked as such.

The relationship model.dal.DatabaseConnection has with model.dal.BoatRepository and model.dal.MemberRepository is using the wrong arrow. It should be a non-striped arrow with a pyramid head.

**Sequence diagram**

The view verbose sequence diagram is well made and shows how the code works. As a developer, this diagram helps greatly.

The edit sequence, sequence diagram is unfortunately named for two reasons. One, it is a sequence diagram so saying "edit sequence, sequence diagram" sounds weird. Two, there are more than one "edit" action in the application (edit member, edit boat). A better name for this diagram would be "Edit Member".

Furthermore, there are some errors in the edit sequence diagram. View.Member does not call a method named “GetIdToEdit”, it reads it from a GET variable. Another minor issue is that the GetUserById() method call should be GetUserById(id).

Also I am having great trouble following this diagram. I can find everything just fine but I do not see it happening in the order shown in the diagram. It seems to be missing steps. For example, c.Program only sends EditMember() to view.Member after going through its own EditMember() method with a “HasEditedMember()” being false. According to the diagram, this is the first step to editing a member. After the user to be edited has been found in the repo, it would seem according to the diagram that the EditMember() method is called again, this time “HasEditedMember()” being true leading to the other path in the if statement. How does this happen? I cannot tell with the help of the sequence diagram.

**Architecture**

Code is divided according to the MVC pattern.

The models are not coupled to the UI in anyway. This means that the UI can be changed without changing the models, which is good. The models do not return formatted strings either, which is also good.

The UI seems to have some domain rules. For example: view.Member has a method called “ValidateMember”. This is not a view job, it is a model or controller job. The view could still output the validation message however. [1, p312]

It seems that only the UI handles output to the console, which is good!

**Unique member ID**

The code creates a unique member ID for each member and it works well.

**Code Quality**

The code follows traditional code indentation standards and is easy to read and understand.

Good names are used for methods and variables. However, some methods are not starting with a capital letter. This can be seen in, for example, view.Member. This is confusing as some methods has it and some do not.

A quick look on the code did not reveal any code duplication.

Due to my editor not showing references from methods in PHP, I cannot (in an easy way) tell if any unused code is present. Do make sure there is none though ☺

**Design Quality**

Code is connected with associations and not IDs in all cases but one; the relationship between model.Member and model.Boat is strictly using IDs.

Code and design follows GRASP, as each class has a responsibility. It does not follow GRASP in the case of the view having domain rules however. [1, p312]

All classes seems to have high cohesion, which is good. I also think the application has low coupling.

The application uses several static (and constant, which in PHP is basically static in other languages) variables.

I could not find any hidden dependencies in the code, which is good.

Information seems to be encapsulated.

The code is defiantly inspired by the domain model. The classes uses the same names that could probably be found in the domain model as well, such as “Boat” and “Member”.

**Overall**

The code itself is very well done and works excellent. However, the diagrams needs some work.

Strong points:

- Code quality
- Overall architecture
- View sequence diagram
- Overall design quality

Weak points:

- Class diagram
- Edit diagram
- View classes using domain rules
- Methods sometimes not starting with a capital letter
- model.Boat and model.Member is only connected with IDs

Because of the problems with the diagrams and the code problems discussed above I do not think the group has passed grade 2 for this workshop. Fix the diagrams and the code however, and you guys will have no problem passing grade 2.

# References:

1. Larman C., Applying UML and Patterns 3rd Ed, 2005, ISBN: 0131489062