Peer Review

Hatem Houssein hh222ix | Alex Spottka as223th | Austin Ponten ap222ti

## **Architecture**

**The application is divided according to the MVC model, however the functions in the controller class “RegistryHandler” seems a bit too complex. There are actual methods which assigns values to variables, which a controller is not meant to do. These things should be done in the model. The controller should, by collaboration with other objects, controll the flow of the application. *(Larman, 39.4)* One could say that currently the “RegistryHandler” is in reality a model class “disguised” as a controller.** (Sources: see links at bottom of document\*)

**Also in the view class “Program” the method delete\_Boat() talks directly to the model Member class :**

**Member m = handler.getMember(personal\_id); This is not allowed and overrides the MVC architecture, coupling the model directly with the UI. *(Larman 13.7)***

**The implementation is created for a console application, limiting the User Interfaces where it can be run.**

## **Source Code Quality**

**The application appears to follow the code standards for C# and upholds the quality of naming since the classes are named in singular and the variables all begin with lower case letters.** (Sources: see links at bottom of document\*)

**The two methods in Program, edit\_Last\_name() and edit\_First\_name(), all prints out practically the same lines of text to the console after the initial question have been asked. This could be derived into a new method which can be used by many edit-methods in the class, to avoid duplicating code.** (Sources: see links at bottom of document\*)

**Since the code is not finished and not runnable it’s very difficult to tell, but it seems like the functions**

**RegistryHandler: public Boat getBoat(string personalNumber, string boatName), RegistryHandler: public void clear(), Member:** **public void addBoat(Boat boat) all have zero references and are so to say “dead”. Dead code is generally considered undesirable, since it for example occupies unnecessary memory and can make the program maintenance more difficult.** (Sources: see links at bottom of document\*)

## **Design Quality**

**The model class Member is the only class that at the moment makes use of any kind of Id, making the application fulfil the requirement of a unique member Id.**

**The application have a high cohesion between its classes which all hold a reasonable size and only handles code within the class’ own responsibility. *(Larman, 17.14)* The classes also have low coupling and are not too connected to other entities within one another. *(Larman, 17.12)***

**The entire Program class is filled with static methods – this should be avoided. Static variables and functions are meant to hold things such as search and file paths which are used several times throughout a class or application, which would cause crashes or fatal errors if they were to be changed. These methods are used once, which means that they have no need to be static.**

(Sources: see links at bottom of document\*)

**The code actively uses getters and setters to reach private encapsulated variables, which is good.** (Larman, 13.7)

**The application does not use primitive datatypes that could be mistaken for complex domain concepts.** (Larman, 9.16, fig. 9.22)

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

**The class diagram would help me in the sense that it shows the total amount of classes in the application, their names, methods and variable types. It does not help me with deriving the structure of the application since it lacks relations, multiplicity, dependencies or any sort of connections between them.**

**The sequence diagram would help me since it shows the process of both main and alternative flows of a use case. It wouldn’t help me since it shows both “View” along with “RegisterHandler”, “Member” and “XML-file”. View is not a class, Member and RegisterHandler are, and XML-file is a persistent storage, not a class. Mixing the terms together makes for a confusing diagram; stick with just classes, since that’s what’s relevant in a sequence-diagram.**

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

**Since the code was unable to compile through the .exe file or even open in Visual Studio it’s very hard to say or find the good points of the implementation in itself.**

**It’s good that the application follows the MVC separation architecture, even though the class RegisterHandler doesn’t follow the exact rules for a controller, as mentioned above.**

**It should be noted that the creators of this application knows and have informed us through the README file that the application is not finished, which makes the peer review much easier to write since it becomes less about finding errors and more about giving suggestions.**

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

**The controller should only delegate tasks to either views or model classes, therefore RegisterHandler should be moved into the model and another, general controller should take its place.**

**It would also be a good idea to give both Member and Boat one controller of their own to better distinguish from where and when the different tasks to the two model classes are delegated.**

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

No. Since the project is not finished it does not pass the grade 2 criteria.

**\*Sources**

**Overall Source:**

Applying UML and Patterns:

An Introduction to Object-Oriented Analysis and Design

And Iterative Development, Third Edition

By Craig Larman

ISBN : 0-13-148906-2

**Links on static variables/methods:**

<https://r.je/static-methods-bad-practice.html>

[*http://blog.schauderhaft.de/2013/07/07/why-static-is-bad-and-how-to-avoid-it/*](http://blog.schauderhaft.de/2013/07/07/why-static-is-bad-and-how-to-avoid-it/)

**Link on dependencies:**

[*http://tutorials.jenkov.com/ood/understanding-dependencies.html*](http://tutorials.jenkov.com/ood/understanding-dependencies.html)

**Link on controller:**

*(*[*https://msdn.microsoft.com/en-us/library/ff649643.aspx*](https://msdn.microsoft.com/en-us/library/ff649643.aspx)*)*

**Link on code standards:**

[*https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp3\_OOP.html*](https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp3_OOP.html)***, Nanyang Technological University***

**Link on duplication:**

[*http://scg.unibe.ch/archive/papers/Rieg98aEcoopWorkshop.pdf*](http://scg.unibe.ch/archive/papers/Rieg98aEcoopWorkshop.pdf)

**Link on dead code:**

[*https://developers.google.com/java-dev-tools/codepro/html/concepts/maintopic*](https://developers.google.com/java-dev-tools/codepro/html/concepts/maintopic)