Report for Habba

Grupp 15

Johannes Gustavsson, Nils-Martin Robeling, Li Rönning,
Alexander Selmanovic, Jian Shin, Camilla Söderlund

Objektorienterat Programmeringsprojekt TDA367
Chalmers tekniska högskola
Sverige
October 2018

1 Peer Review

Do the design and implementation follow design principles?
Does the project use a consistent coding style?
Yes, small classes, small methods for the most part, with exceptions for some bigger classes with methods that could be broken down into smaller parts. For example a lot of if-statements in a row in RegisterPresenter

Is the code reusable?
Some methods would be more reusable if they were decomposed into smaller methods. There are methods returning interfaces which lowers dependencies.

Is it easy to maintain?
Would be easier if some methods used more functional decomposition. But for the most part yes.

Can we easily add/remove functionality?
Uses interfaces and factories for abstraction and extendability. The code can be adjusted to use a real database instead of the internally stored files that are used now. Both this, and the fact that the user password is not encrypted is reasoned for in the SDD. (not really necessary for this project)

Are design patterns used?
Singleton pattern is used in the project as described in the project SDD. MVP is used, for more information see "Is the code easy to understand? Does it have an MVC structure?" Iterator and Adapter pattern was found when the project was read on first glance. An example of this can be seen in the class EventCollection where an iterator is used to iterate over the different collection of events.

Is the code documented?
The amount of comments varies between different parts of the code. Some classes and methods are well documented while others are still missing their documentation. This will probably be added in the future.

Some of the comments can be seen as unnecessary when it comes to commenting getters and setters since these in general have a obvious purpose. This makes the code less readable on first glance.

Are proper names used?
Proper name are mostly used in the project, which means that when reading the name of methods/classes/interfaces it is possible to understand what that part of the project is supposed to do without looking at the code.

The definition of "event", "task" and "deadline" is confusing. "Task" and "deadline" is defined as the same as "event" when in fact "task" should be something to be done and "deadline" is when to do it. This is written in both

the SDD and as a documentation in the class Event.

In some of the classes, for example MonthCalendarPresenter, the local variables in some methods have very inexplicit names so that it is more difficult to understand what is happening.

Is the design modular?

The design is partially modular, it is packaged in the larger groups: (model, presenters, services and views). It might be even more modular if the classes would be grouped by the class's area of use, for instance could everything connected to 'Event' in the model-package be put in a smaller package to be even more modular.

Does the code use proper abstraktions?

The code uses proper abstractions, which can be seen in the code with the implementations of a lot of different interfaces.

Is the code well tested?

The code has some tests connected to it, but since we cannot build the project it is impossible to see if they pass or not. A lot of classes and bigger methods are missing tests.

Are there any security problems, are there any performance issues?

There is some security for example ZxcvBn. They don't explain what Zxcvbn is or does. When first viewed it is very confusing and required Google to understand.

Is the code easy to understand? Does it have an MVC structure?

They have used MVP pattern which seems to be followed. It could be further explained in the SDD so that people with little to no knowledge about the pattern could understand what it is about.

Can the design or code be improved? Are there better solutions?

Yes, overall it's very good, but some things could be implemented for clarification and easier overview. For example more functional decomposition in larger methods in combination with some documentation of the external libraries used in the project.

Well commented code in Model package.