Do the design and implementation follow design principles?

- **Does the project use a consistent coding style?**

  Yes, the code style is consistent. Method, class and variable names follow the same patterns as an example.

- **Is the code reusable?**

  The “actual” model in the program is made up of the ViewModel classes which also handle certain view aspects of the program. This means that the model/code is not reusable as there would be problems if an attempt was made to use it in a different program.

- **Is it easy to maintain?**

  Since there are no tests, it might be difficult to maintain the program in the case of bugs appearing for example.

- **Can we easily add/remove functionality?**

  Adding functionality shouldn’t be too difficult. There are no strong dependencies between existing Model classes and adding new ones should be fairly straightforward. For every new view though, you would have to add a method for switching to it, in all of the existing controllers.

- **Are design patterns used?**

  The ones we found were MVVM/MVC pattern and the Observer pattern.

Is the code documented?

There is barely any documentation in the code, at least none we could find.

Are proper names used?

The names of the classes are confusing at first glance. It’s not really clear what TimerViewModel, as an example, is meant to do. Some classes are also in plural, when they should be in singular.

However, the names for variables and methods are very clear and easy to follow. In some places there are several methods and variables with very similar names though. The clearest example is in ToDoViewModel where there are methods called addToDoList and addToDoLists as well as the variable allTodoLists.

Is the design modular? Are there any unnecessary dependencies?

We couldn’t find many, if any, unnecessary dependencies. There are however many public methods that could lead to dependencies if one isn’t careful. Most variables are package-private or private.

Does the code use proper abstractions?

No. The only abstraction is the abstract Controller class, but it is only implemented by HelpViewController and the method inherited by Controller is not used.

Is the code well tested?

No, there are no tests.

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

None that we could find. There could possibly be issues in the future with the creation of new controller instances every time a view is switched (more below).

Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

The code has a structure that is a mix of MVC and MVVM. There are Views, Controllers, ViewModels and Model classes. This combination makes the code quite hard to understand, and even more so since there is only one package (for the controllers) and the model isn't isolated from the other parts structurally.

The Model classes consist mostly of data, and it's the ViewModel classes that handle the logic. According to the SDD, MVVM is being used but then there should be no Controller classes and the ViewModel classes should only wrap the model and provide observable data for the Views to listen to. Currently the ViewModel does provide observable data but it also handles logic as mentioned, which is not it's responsibility.

An example of this is when a ToDoList is added. The input is handled by the Controller (correct) which then passes the information to the ViewModel which creates an instance of the ToDoList class and saves it (incorrect, should be done by the model). The ViewModel also is the one that updates the view here, which it shouldn't do either.

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

Every time a user switches between views, a new controller instance for that view is created (by the creation of a new FXMLLoader). This means that, for example, when you exit and re-enter the ToDoListView, the ToDoListViewModel has been instantiated and none of the saved ToDoLists remain.