Lab exercise – Design for Testability

In this exercise, you will use some of the techniques you have learned to *design for testability*. Specifically, you will be acquainted with the identification of *dependencies* and introduction of *interfaces* to lower the couplings in your design. This will train you to design for testability and to refactor existing code for testability.

The scenario is that you are provided with some legacy code for an ECS system, which is *not* particularly well-designed. Neither is it documented other than by the occasional source code comments. This is not because your teacher is lazy – it is a very, very common scenario in the industry, believe me!

Your *first* job is to investigate the legacy code. Your *second* job is to do some extensions to the code.

You are required to demonstrate both techniques for dependency injection (constructor injection and property injection).

Exercise 1: Set up the team's infrastructure
Have 1 person in the team commit and push the ECS Legacy Solution available from Blackboard to a new empty Git repository on GitHub.Then, have everybody in the team else clone the repository. At this point, all team members have a clone of the legacy code.

Exercise 2: Reverse engineering the legacy ECS solution
Reverse-engineer the ECS legacy code: Create a UML class diagram of the existing solution and use this to identify the flaws in the design when it comes to testability.

Exercise 3: Implement the refactored design
Exercise 3.1:
Create a design that, for each of the flaws you identified before, proposes a solution. Apply the techniques discussed in class to make the design more testable.

Exercise 3.2
On team member 1's PC: Implement your refactored design, in a new application project, under the same solution (which is already version controlled from the previous steps). Commit and push the changes.

Exercise 3.3
On team member 2's PC: Pull the changes. Then, create unit tests for the refactored version of the class ECS, in a new test class library project.

To do this, you will need fake "versions" of the dependent classes. Consider where these fakes should be placed in the solution – in the *application* project or the *test* project?

Consider what you try to test for. What should the various fakes do for you to be able to test that?

Exercise 3.4
Set up a Jenkins project on the CI server for your solution and ensure it runs your unit test on your project.

Exercise 4: Extend the ECS
You are now going to extend the ECS with two more features. As you create your design, be sure to design your changes for testability, implement your changes according to the design, test your changes and push them to the git repository so that all team members – and Jenkins – are in sync. Also, be sure to identify and handle any exceptions that may occur by the design extensions.

Extension 1: Add a window to the ECS. The window should open when the temperature rises above a certain upper temperature threshold (different from the heating threshold) and close if the temperature is below this threshold. Of course, it doesn't make sense, if the heater can be on and the window open at the same time.

Extension 2: Extend the threshold mechanism to be configurable so that they can be changed at run-time, and it must be checked that they are consistent, when set.