Self Study 1 – TDD Coding Kata “Help ALF“

Help ALF
Late last night in the Tanner household, ALF was repairing his spaceship so he might get back to Melmac. Unfortunately for him, he forgot to put on the parking brake, and the spaceship took off during repair. Now it's hovering in space.
ALF has the technology to bring the spaceship home if he can lock on to its location.

Given a map:

```
..........
..........
..........
.......X..
..........
..........
```

The map will be given in the form of a string with \n separating new lines.
The bottom left of the map is [0, 0]. X is ALF's spaceship.

In this example:

findSpaceship(map) => [7, 2]

If you cannot find the spaceship, the result should be
"Spaceship lost forever."
Can you help ALF?

1. Take the String Calculator Kata as base framework
2. Make a Git repository with all files (Todo, src, spec)
3. Commit every single TDD step (1,2,3) with an appropriate commit message
4. First step is writing the Todo List
5. Then start with TDD step 1 write failing test for first todo, commit
6. Continue with TDD step 2 write simplest production code, commit
7. Refactor if necessary (step 3) and commit
8. Repeat from step 5 for every Todo in the list
9. Upload the zipped GIT repository in Moodle

Self Study 2 – TDD Pair Programming Coding Kata “Electrons around the cores“

Electrons around the cores

Have you heard of the game "electrons around the cores"? I'm not allowed to give you the complete rules of the game, just so much:

The game is played with 4 to 6 dice, so you get an array of 4 to 6 numbers, each 1-6
The name of the game is important
You have to return the correct number five times in a row and your solution is considered to be correct

If you just call "submit", you'll get an array and the expected value!

Here are some input/output pairs for you to wrap your mind around:

[1, 2, 3, 4, 5] -> 6
[2, 2, 3, 3] -> 4
[6, 6, 4, 4, 1, 3] -> 2
[3, 5, 3, 5, 4, 2] -> 12

Build a pair group with one of your colleagues and use Pair Programming TDD to develop the solution.

1. Take the String Calculator Kata as base framework
2. Make a Git repository with all files (Todo, src, spec)
3. Commit every single TDD step (1,2,3) with an appropriate commit message including who was doing this step
4. First step is writing the Todo List
5. Then start with TDD step 1 write failing test for first todo, commit
6. Continue with TDD step 2 write simplest production code, commit
7. Refactor if necessary (step 3) and commit
8. Repeat from step 5 for every Todo in the list
9. Upload the zipped GIT repository in Moodle

Self Study 3 – TDD Legacy Code Coding Kata "Gilded Rose"

How to use this Kata

The simplest way is to just clone the code and start hacking away improving the design. You'll want to look at the **GildedRoseRequirements.txt**, which explains what the code is for. I strongly advise you that you'll also need some tests if you want to make sure you don't break the code while you refactor.

You could write some unit tests yourself, using the requirements to identify suitable test cases. I've provided a failing unit test in a popular test framework as a starting point for most languages.

Whichever testing approach you choose, the idea of the exercise is to do some deliberate practice, and improve your skills at designing test cases and refactoring. The idea is not to re-write the code from scratch, but rather to practice designing tests, taking small steps, running the tests often, and incrementally improving the design.

You are allowed to use AI for support of single steps. If you use AI support, mark it in the commit message and shortly describe the kind of support.

1. Choose the language you like
2. Make a Git repository with all files of the language you chose (src, spec)
3. Study the requirements and try to understand the legacy code
4. Write unit tests to cover all requirements in the legacy code
5. Commit every single test case with an appropriate commit message as soon it runs green
6. Create a failing test for the new requirement (see file), commit
7. Implement the new requirement, the failing test should be green, commit
8. Refactor the whole legacy code and rerun all tests, should be green
9. Commit the final refactored solution with all tests green
10. Upload the zipped GIT repository in Moodle

Self Study 4 – Static Code Analysis or Consumer Driven Contract Testing

Choose, which example you would like to provide…

Static Code Analysis

Use an existing project (one of your projects or choose a Github repository of an application you like). Make an own git repo with all the source files.

Install and configure a source code analysis tool of your choice and take some screenshots. Run the source code analysis in your IDE against the chosen source code and document the 10 most important findings with screenshots. Correct the 10 most important findings in the code. Commit the code. Upload a zip containing the screenshots and the git repository to Moodle.

Consumer Driven Contract Testing

Use an existing project dealing with REST-Services (one of your projects or choose a Github repository of an application you like). Make an own git repo with all the source files.

Install and configure pact.io, so that you can access the contract server. Define contracts for the services in your code. Commit the code and the contracts. Then seed 3 "bugs" in the consumer code or the provider code (if your code does not include consumer calls, write an artificial consumer). Run the consumer and the provider tests and check, if the seeded bugs are found. Capture the logs and screenshot of the tests run. Upload a zip containing the logs, screenshots and the git repository to Moodle.