

AU ENGINEERING

I4SWT MANDATORY EXERCISE

AIR TRAFFIC MONITORING

TEAM 16-1-6

Name	Student ID	E-mail
Ao Li	201407737	liao0452@gmail.com
Cecilie Bendorff Moriat	201405949	ceciliemoriat@gmail.com
Jonas Møgelvang Hansen	201407199	jonas_jmh@gmail.com
Morten Sand Knudsen	201270955	mortensandknudsen@gmail.com

CI BUILD JOBS

Unit tests:

`http://ci1.ase.au.dk:8080/job/Team%2016-1-06%20ATM%20(Unit%20Test)`

Integration tests:

`http://ci1.ase.au.dk:8080/job/Team%2016-1-06%20ATM%20(Integration%20Test)`

Code metrics:

`http://ci1.ase.au.dk:8080/job/Team%2016-1-06%20ATM%20(Code%20Metrics)`

APRIL 21, 2016

Contents

Contents	2
1 Introduction	3
2 Design	3
2.1 Design considerations	3
2.2 Implementation	3
3 Test	5
3.1 Unit Test	5
3.2 Integration Test	5
3.3 Jenkins	6
4 Teamwork	6
4.1 Strategies	6
4.2 Continuous integration	6
5 Conclusion	6

1 Introduction

The purpose of this journal is to reflect upon the design, implementation and test of the Air Traffic Monitor system (ATM).

The exercise required not only a working system, but a special effort had to be made to obtain a generic design with an appropriate amount of tests which should be simple to maintain if changes in the exercise requirements were to be made.

2 Design

As earlier stated the design of this solution was given thought as it had to be extensible and adaptive to changes in requirements. This section describes the process of obtaining such design and the outcome of the reflections.

2.1 Design considerations

An effort were made to design the system based on the five basic principles of object-oriented programming and design, SOLID. These principles applied to a system tend to make this maintainable and extendable.

To further create abstraction from the provided .DLL and follow the Dependency-Inversion principle, the ATM makes use of a modified repository-pattern¹. This allows for a quick change in the data-source to a database for example. The modification only allows for reads in the repository, the implementation can be seen in the 'IReadOnlyRepository<TModel>' interface.

Two sequence diagram for the back-end can be seen in figure 0.1 and 0.2. Where the repository-pattern is used to connect the monitor to the datasource, in this case the provided .dll.

2.2 Implementation

It works

¹<https://msdn.microsoft.com/en-us/library/ff649690.aspx>

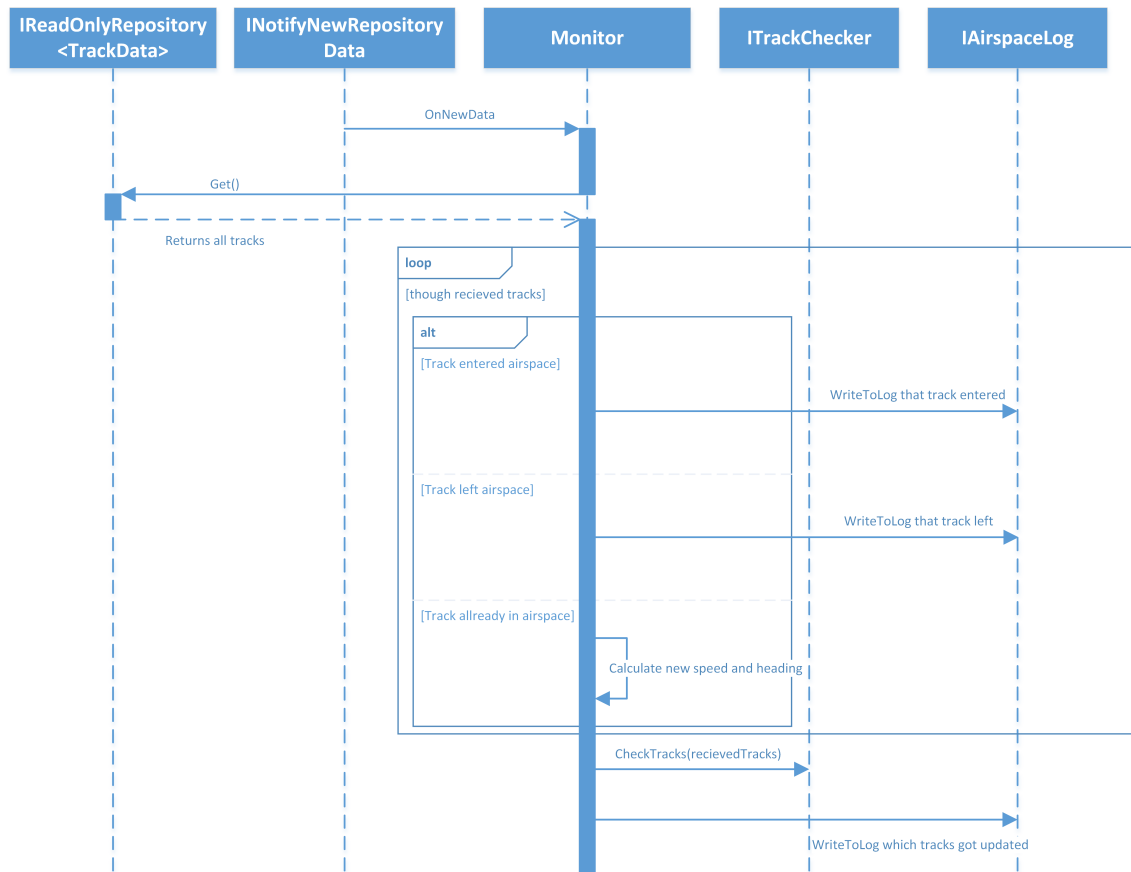


Figure 0.1. Sequence diagram for monitor

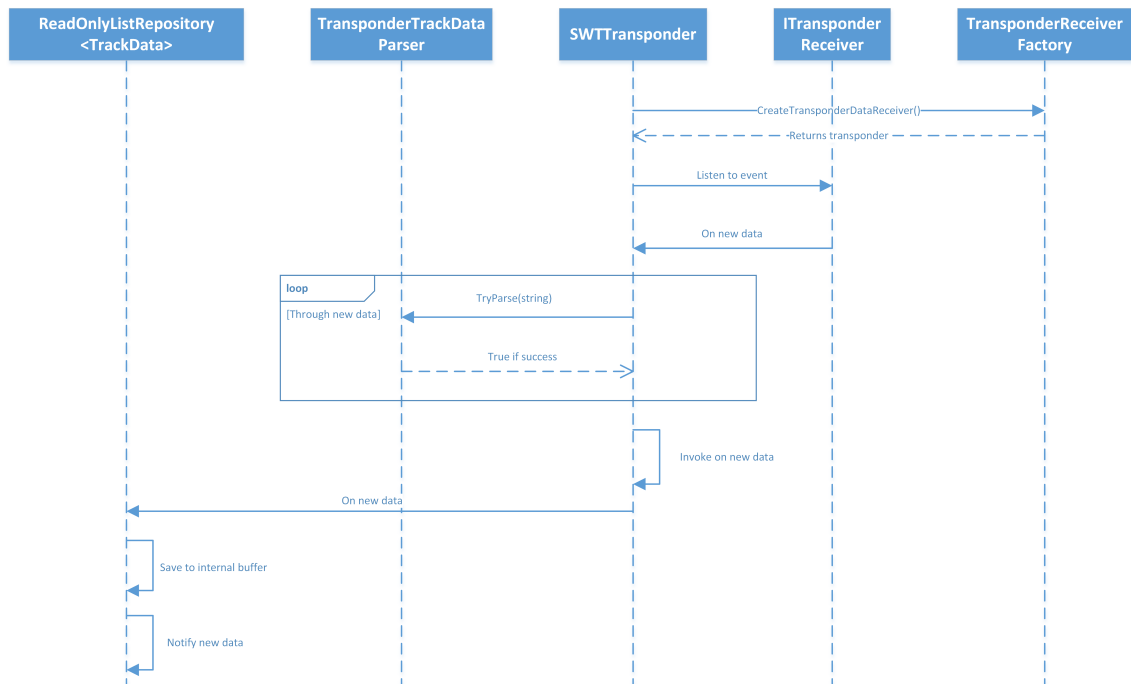


Figure 0.2. Sequence diagram for data acquisition from the transponder .DLL

3 Test

This section describes how testing supported the software development of the ATM system.

3.1 Unit Test

The software development was closely followed by unit tests to ensure the code reacted as intended and as the requirements demanded.

As reviewed in the I4SWT course multiple practices for writing unit tests exist. In this assignment both *Test Driven Development (TDD)* and *Feature Driven Development (FDD)* were used.

TDD is an agile practice where test should be written before the implementation. This ensures that no code which has not been tested is written. The development cycle tend to be short and the practice encourages simple design in the code.

FDD is the opposite of TDD in where the implementation is written before the tests. Thus the functionality is in focus and the programmers can later decide the satisfying amount of tests. This practice was used more extensively than the TDD as it is less time consuming to implement the features. A drawback was that the focus does not lay with the tests, which could be written at such late time where finding a bug could be a larger inconvenience, than if it was found during the writing of the feature.

3.2 Integration Test

Integration testing is the process where the individual software modules are combined and tested as a group. This is to be done after the unit tests have passed.

Preparation for integration testing lies in documenting the modules in a dependency tree as shown in figure 0.3.

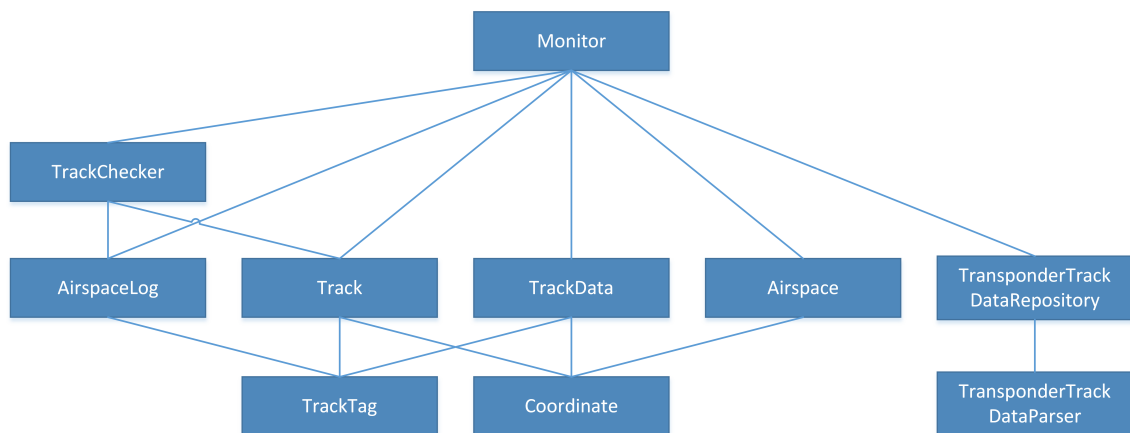


Figure 0.3. Dependency tree for ATM

The Dependency tree shows the integration and interdependence of our different classes. We have chosen to use a bottom-up integration. The choice behind this is that a lot of the integration test lies in the unit test for the ATM.

3.3 Jenkins

Jenkins is a program that ensures CI on our repository. When committing code to the repository Jenkins will first run a build that runs all the unit tests, if this succeeds, Jenkins will subsequently run a project which tests all integration test. Once approved it will finally run a project that calculates the code metrics for the project.

4 Teamwork

In this section the teamwork is described. The requirements were 3-4 people in a group and no more than two persons should share a computer while programming. Another requirement was the use of *Continuous Integration*, which helps the developers commit to a shared build server multiple times during the development process.

4.1 Strategies

Earlier experience from working with some of the strategies from *Extreme Programming* in the course I4SWD, led to the assumption that it could be suitable to use for the software development of the Air Traffic Monitor too.

One of these strategies was *Pair programming*. Code is then written by pairs which share the workstation. One will be in control of the keyboard and write the code while the other will watch the code and work towards the best implementation. The pair switches place every now and then. This ensures that both programmers are engaged in the software.

4.2 Continuous integration

As the group was divided into two pairs of developers each working on classes of their own, the continuous integration helped the two groups to gain a shared understanding on the software development progress.

Another benefit was the automatic generated code coverage report and software quality metrics which were used to determine whether the written software and tests were satisfying. If not, it was easy to gather information where the code standards should be optimized for better statistics.

As with every other git project a version history was obtained making it simple to revert to a previous build if changes caused a broken build.

5 Conclusion

The process of developing the Air Traffic Monitoring system and compile this journal was made in two weeks. During these weeks multiple strategies from the courses I4SWT and I4SWD were used to satisfy the requirements given.

At all times during the design, implementation and testing, it was ensured that every module would be following the SOLID principles used in agile development. It was given

that the requirements could be changed at a time which potentially could cause a major need for re-implementing the system.

Even though it might have taken longer to develop the ATM knowing the possibility of a requirement change, it worked as a great way to motivate reflections upon the design of the system. Furthermore being able to try the different unit testing practices on a larger scale and compare these has been interesting.