

SWT - Handin 3 Gruppe 5

Members

Navn	E-mail	Studienummer
Danny Tran	au556770@uni.au.dk	201610981
Bekir Karaca	au555204@uni.au.dk	201610458
Muhanad Al-Karwani	au557732@uni.au.dk	201611671
David Lo	au516587@uni.au.dk	201404972

Github URL
https://github.com/D4V1DD33P/SWT5_ATM_DEL2

Jenkins Static Analysis
http://ci3.ase.au.dk:8080/job/5_SWT_ATM_DEL2_StaticAnalysis/

Jenkins Coverage
http://ci3.ase.au.dk:8080/job/5_SWT_ATM_DEL2_Coverage/

Jenkins SQM
http://ci3.ase.au.dk:8080/job/5_SWT_ATM_DEL2_SQM/

Jenkins Integrationtest
http://ci3.ase.au.dk:8080/job/5_SWT_ATM_DEL2_Integrationtest/

Indhold

1	Introduktion	3
2	Systemkrav	3
2.1	Spor gengivelse	3
2.2	Event detektering, logging & gengivelse	3
2.2.1	Detaljer omkring Seperation event	4
2.2.2	Detaljer omkring "Track Entered Airspace"-hændelser	4
2.2.3	Detaljer omrking "Track Left Airspace"-hændelser	4
2.3	Arbejdsfordeling	4
2.3.1	Contributors	5
2.4	CI	5
3	Design	6
3.1	Fremgangsmåde	6
3.2	Klassediagram	6
3.3	Sekvensdiagram	8
4	Implementering	8
5	Dependency tree	8
5.1	Integration test	9
6	Static Analysis	9
7	Resultater	10
8	Diskussion	10
9	Konklusion	10

1 Introduktion

I denne Hand-in videreudvikler vi sidste Handin, som var et Air Traffic Monitoring System. Den skal kunne gøre det samme som sidst, med nogle ekstra udvidelser.

2 Systemkrav

2.1 Spor gengivelse

Selve systemet skal kunne overvåge et bestemt område, hvor hver fly i området bliver udskrevet på en terminal. Fly der ikke er i det bestemte område skal ikke udskrives.

Området der bliver overvåget har sine grænser, med minimum og maksimum koordinater og højde.

- Den sydvestlige del af området har et koordinatsæt på (10.000, 10.000).
- Den nordøstlige del af området har et koordinatsæt på (90.000, 90.000).
- Højdens minimale grænse er 500 meter, samtidig med at højdens maksimale grænse er 20.000 meter.

Flyene der bliver registreret i området skal udskrives med følgende informationer:

- Tag (Tekststring, 6 karakterer)
- Nuværende position (x-y, begge i meter-enheden)
- Nuværende højde (meter)
- Nuværende hastighed (m/s)
- Nuværende kurs (0-359°, med uret, 0° er nord)

2.2 Event detektering, logging & gengivelse

- Når transponderen detekterer data, skal dataen undersøges for en hændelse af forskellige events.
- Alle hændelser af et "seperation"event skal skrives til en konsol med følgende data (Det kan formodes, at hændelser ikke gengives på skærmen):
 1. Tidspunkt for hændelsen
 2. Et Tag af de involverede spore.
- Alle nuværende events skal gengives på skærmen så længe de er aktive.
- Det er kun nuværende events der skal gengives på skærmen, hvilket vil sige, at der ingen historik over events er.

2.2.1 Detaljer omkring Separation event

- Hvis den vertikale luftrum mellem to spore er mindre end 300 meter , og den horisontale luftrum er mindre end 5.000 meter på samme tid, skal sporene anses som at være i en "konflikt". Dvs. at når to spore er i en konflikt skal det gengives på skærmen.
- Et Separation event skal blive ved med at gengives på skærmen så længe de to spore stadig er i konflikt med hinanden.
- Når de to spore ikke længere er i konflikt, skal den tilhørende Separation event deaktiveres.
- Gengivelsen af Separation eventet skal inkludere Tags af de involverede spore

2.2.2 Detaljer omkring "Track Entered Airspace"-hændelser

- Når et nyt spor flyver ind i det overvågede luftrum, skal et "Track Entered Airspace"-event aktiveres.
- Et "Track Entered Airspace"-event skal forblive aktiveret i 5 sekunder.
- Gengivelsen af "Track Entered Airspace" skal indeholde de spor der er involveret og tiden af hændelsen.

2.2.3 Detaljer omkring "Track Left Airspace"-hændelser

- Når et nyt spor forlader det overvågede luftrum, skal et "Track Left Airspace"-event aktiveres.
- Et "Track Left Airspace"-event skal forblive aktiveret i 5 sekunder.
- Gengivelsen af "Track Left Airspace" skal indeholde de spor der er involveret og tiden af hændelsen.

2.3 Arbejdsfordeling

Arbejdsfordelingen blev opdelt efter interesse, og hvad der har givet mest mening i forhold til styrker og svagheder hos gruppemedlemmerne, så den enkelte gruppemedlem kunne få mest ud af arbejdet. Gruppen har naturligvis hjulpet hinanden igennem hele projektet, men ligeledes har der været plads til individuel arbejde mht. vores tests. Hver gruppemedlem fik ansvaret for at oprette en klasse, et interface samt unit tests og integration test. Dette afspejler sig i de 4 tests, som er blevet udført.

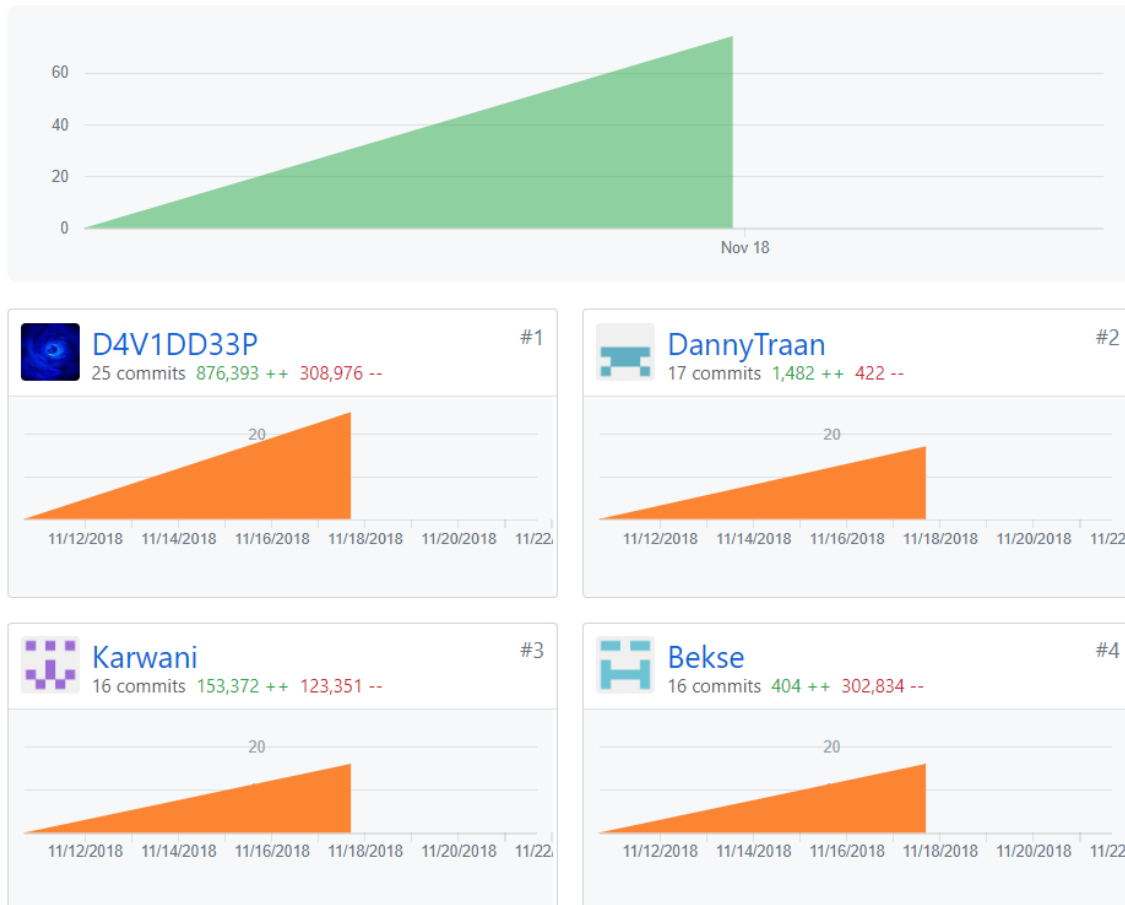
Klassebeskrivelser:

- ParseData (David & Muhanad): Tager List<string> fra .dll filen, og derudfra laves en liste af track-objekter.
- FilterData (Danny): Sikre at den modtagne liste er indenfor gyldigt luftområde. Kalder update-funktionen med listOfTrackInfo, som indeholder de fly, der er inde i luftrummet.
- TrackUpdate (Bekir): Kalkulerer speed og course samt opdatere med ny data.
- DetectVicinity (David & Muhanad): Detekterer nærheden af flyene.

- TrackData (Bekir & Danny): Holder øje med nødvendig data hhv. Altitude, Speed og Course og udskriver disse med korrekt enhed
- VicinityData(David & Muhanad): Gemmer vigtig data til en logfil og udskriver events

2.3.1 Contributors

Nedenfor kan man få et overblik over gruppemedlemmernes bidragelse til projektets repository.



Figur 1: Contributors

2.4 CI

Under udarbejdelsen af projektet er der blevet lavet et Jenkins-projekt på domænen <http://ci3.ase.au.dk:8080/>. Den er blevet anvendt til at holde styr på modultestene og gruppens fremskridt. Yderligere har Jenkins været effektiv når gruppens medlemmer ikke har siddet i samme rum og arbejdet på projektet. Den har sørget for, at man altid har kunnet se ændringer, som hver enkelt medlem har foretaget sig, hvilket vil sige, at den laver en build-test for hver gang der bliver pushed på vores GitHub repository. Det har været en fordel at bruge Jenkins da, vi i denne Handin også skulle gøre brug af Static Analysis, Coverage og IntegrationTest. Disse pipelines har hjulpet os med at finde fejl i koden samt givet et bedre overblik over koden.

3 Design

3.1 Fremgangsmåde

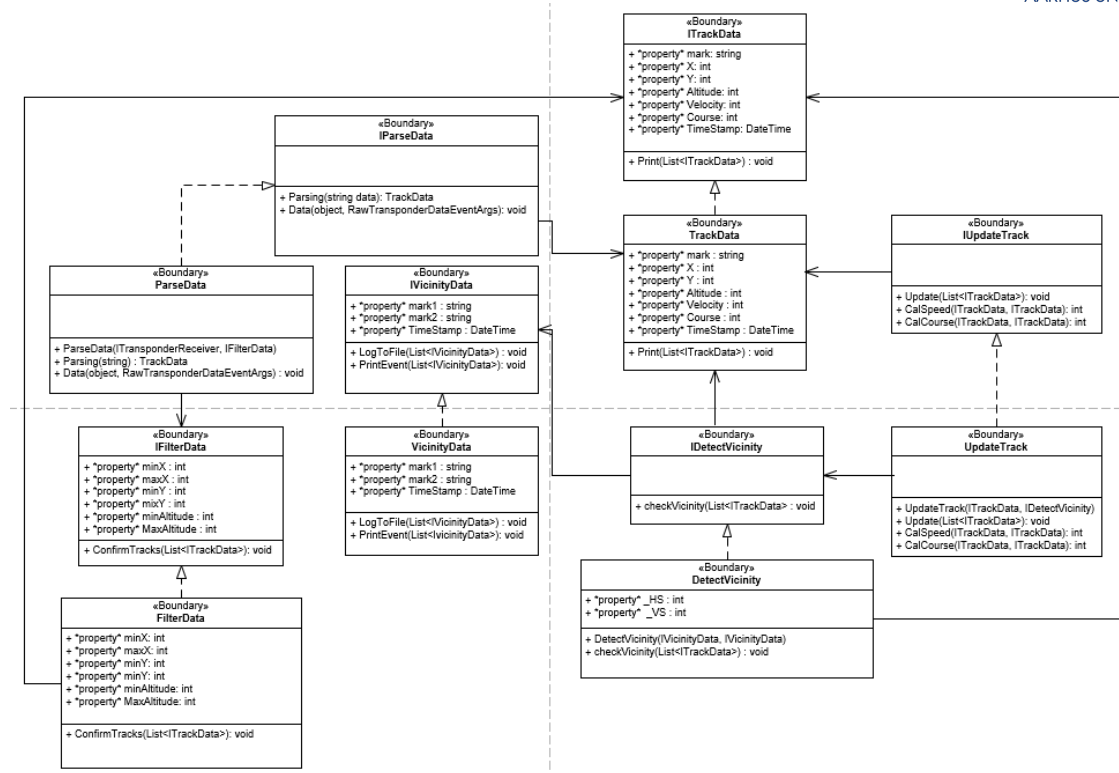
Gruppen er kommet frem til software-arkitekturen ved at gå detaljeret igennem opgavebeskrivelsen. Yderligere har gruppen været i dybden med systemkravene og fundet passende klassenavne samt funktioner til sekvensdiagrammet og klassediagrammet, som ses på Figur 2 og 3. Vores software-arkitektur bærer særdeles præg af måden hvorpå diagrammerne er sammensat og artefakterne deri.

Efter oprettelse af repository for AirTrafficMonitoring(ATM), oprettede vi en solution med 4 forskellige projekter:

1. AirTrafficMonitoring - Indeholder selve programmet.
2. AirTrafficMonitoring.Test.Unit - Indeholder modultest af klassefiler.
3. AirTrafficMonitoring.Test.Integration - Indeholder integrationstest af unit tests.
4. ATMLibrary - Indeholder klassefiler.

3.2 Klassediagram

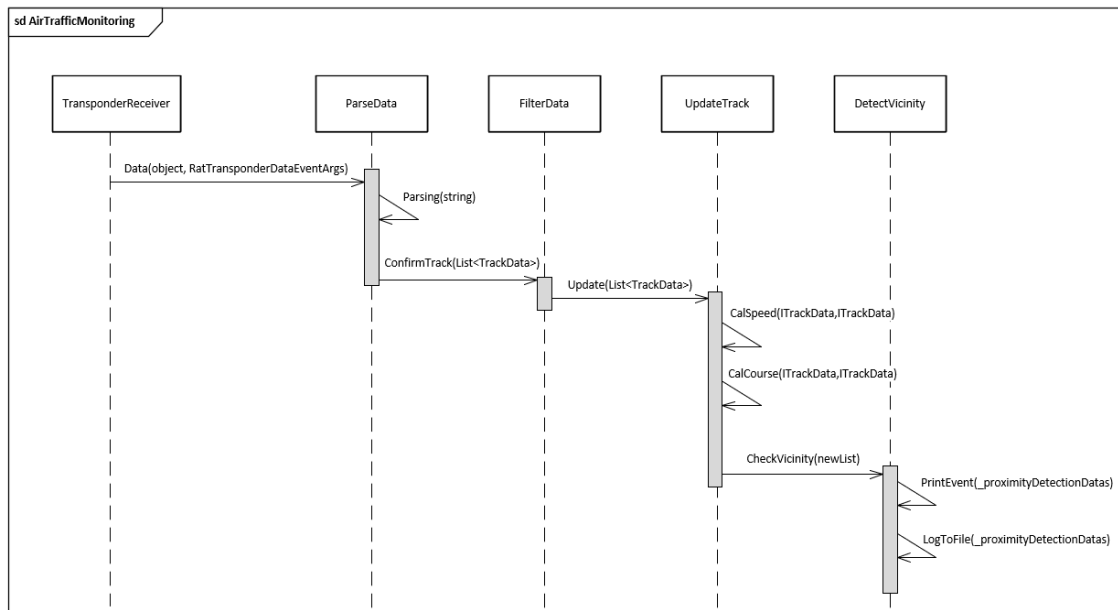
Nedenfor kan man se vores udarbejdet klassediagram, som tager udgangspunkt i opgavebeskrivelsen. Måden hvorpå vi har sat klassediagrammet op på, har vi fra et eksempel på lektionsslide. Vi har i gruppen nøje bestemt hvilke klasser, der skulle medtages. For os virker det mest intuitivt at opstille en løsning som vist på klassediagrammet. Vores klassediagram er præget af højt abstraktionsniveau i og med at vi bruger interfaces, hvilket gøre koblingen mindre mellem klasserne. Yderligere årsag til anvendelse af interfaces er at vi kan implementere fake-afhængigheder uden at det påvirker den rigtige klasse.



Figur 2: Klassediagram over systemet

3.3 Sekvensdiagram

Ud fra klassediagrammet er der blevet udarbejdet et sekvensdiagram for det overordnede systems interaktion med de forskellige klasser, som kan ses nedenfor.



Figur 3: Sekvensdiagram over systemet

4 Implementering

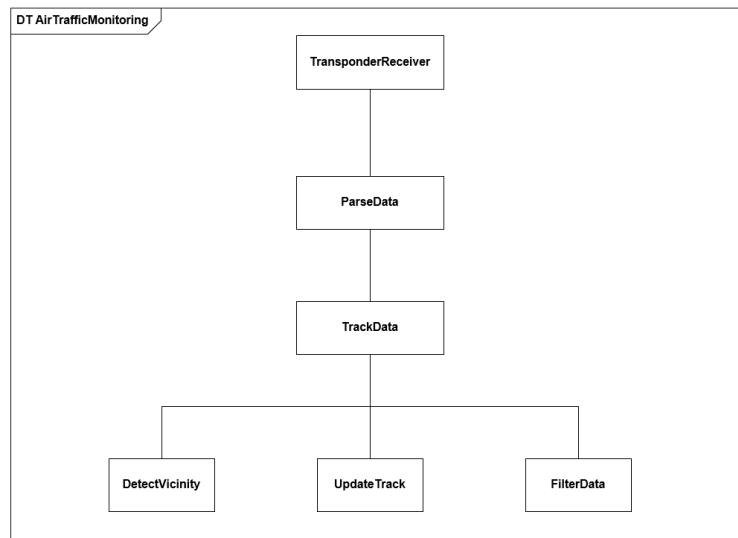
Vi har løbende under implementering af ATM systemet testet applikation via unit testing. Gruppen har valgt at dele testene op i 4 overordnet kategorier, som indeholder flere mindre teste dvs. vi har lavet unit testing.

Her har gruppen specielt haft fokus på at anvende fakes, hvilket kan ses med den mængde af interfaces. Interfaces gør det nemlig muligt for de abstrakte klasser, at ikke bekymre sig for de nedarvede klasser. I forbindelse med fakes har vi valgt at anvende NSubstitute så vi selv slipper for at kode fake klassen.

5 Dependency tree

Her ses gruppens Dependency tree, som har til formål at visualisere afhængigheder i systemet. Med dependency tree ses Arve-hierarkiet ikke. Vi kan udefra vores dependency tree se at høj niveau modulerne ligger i den ene ende og lav niveaue i den anden.

- ParseData afhænger af TransponderReceiver
- TrackData afhænger af ParseData
- DetectVicinity, UpdateTrack og FilterData afhænger af TrackData



Figur 4: Dependency tree af ATM

5.1 Integration test

Det næsten trin gruppen har gribet an er integrationstest, som er en fremgangsmetode, der anvendes til verificering om unit testene virker når de skal implementeres sammen med hinanden. Vi har i denne del testet interaktiviteten og interfaces imellem moduler, som også virker sammen.

Før vi kunne teste integrationstesten skal test af alle vores unit tests være færdige uden fejl(ellers vil vi ikke kunne vide hvor fejlen ligger). Yderligere skal vi have et færdiglavet Dependency tree ift. vores systemarkitektur. Til slut, for at kunne lave håndgribelig integrationstest skal vi have lagt en plan for integrationstesten dvs. hvilken type af integrationstest vi ønsker at udføre. Nedenfor ser man en integration plan hvilket er arbejdet ud fra vores Dependency Tree (DT).

Step	TransponderReceiver	ParseData	TrackData	DetectVicinity	UpdateTrack	FilterData
1			S	X	S	
2		T	S	X	X	X
3	T	X	X	X	X	X

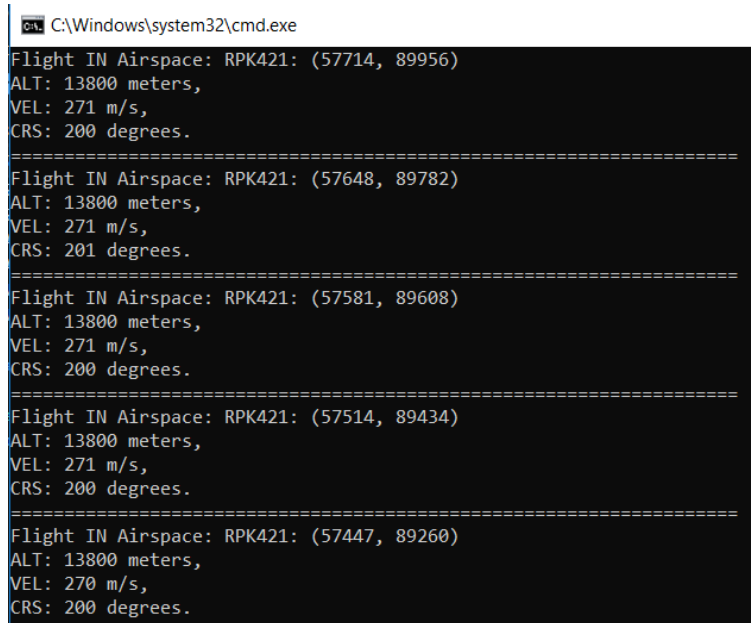
- T: Det inkluderede modul, det er top-modulet.
- X: Dette modul er inkluderet.
- S: Dette modul er forfalsket.

6 Static Analysis

Static Analysis går ud på at debugge noget kode uden at køre selve programmet. Static Analysis er et disciplin, som har det formål, at det giver feedback til programmøren omkring den skrevne kode. Ved at man laver en Static Analysis af koden, får man afsløret nogle fejl, der ikke manifesterer sig. Static Analysis bruges også inde på vores Jenkins server. Da vi har lavet et webhook til vores GitHub, bliver vores Jenkins jobs automatisk kørt hver gang, der bliver pushed op på repository'et.

Inde på vores Jenkins server bruger vi et værktøj der hedder FxCop, hvilket bruges til Static Analysis. FxCop fortæller os hvilke warnings vi får i koden samt fortæller os hvilke warnings, der er blevet fixed. FxCop advarer os om, hvilke warnings der er vigtigst at fikse først.

7 Resultater



```
C:\Windows\system32\cmd.exe
Flight IN Airspace: RPK421: (57714, 89956)
ALT: 13800 meters,
VEL: 271 m/s,
CRS: 200 degrees.
=====
Flight IN Airspace: RPK421: (57648, 89782)
ALT: 13800 meters,
VEL: 271 m/s,
CRS: 201 degrees.
=====
Flight IN Airspace: RPK421: (57581, 89608)
ALT: 13800 meters,
VEL: 271 m/s,
CRS: 200 degrees.
=====
Flight IN Airspace: RPK421: (57514, 89434)
ALT: 13800 meters,
VEL: 271 m/s,
CRS: 200 degrees.
=====
Flight IN Airspace: RPK421: (57447, 89260)
ALT: 13800 meters,
VEL: 270 m/s,
CRS: 200 degrees.
```

Figur 5: Konsolbillede efter kørsel af applikationen

8 Diskussion

Der kunne have været sat mere fokus på Single Responsibility Princippet (SRP), således at der kun var et eller færre ansvar i hvert enkelt klasse. Ideelt er formålet at opnå lavere kobling mellem vores klasser og gøre det lettere at teste klasserne. Yderligere vil SRP have bidraget med et mere letlæseligt tekst. Udover dette har vi ikke formået at lave test-cases for alle funktioner i systemet.

9 Konklusion

Igennem udarbejdelsen af vores løsningsforslag har vi gjort brug af en teori fra undervisningen. Ligesom sidste handin har vi gjort brug af en Jenkins server, til at holde styr på vores unit tests, integrations tests, SQM og Static Analysis. Derudover er ReSharper Ultimate også brugt gennem Visual Studio til at teste vores klasser samt NSubstitute.

På trods af gruppens indsats har vi alligevel punkter, hvorpå vi kunne have forbedret arbejdet. Eksempelvis opstillede vi Jenkins serveren relativt sent. Initialt prioriterede vi at teste vores klasser i Visual Studio med ReSharper løbende i udarbejdelsen.