SWT Aflevering 3

Air Traffic Monitoring

Team number: 3

25/04-2018

|  |  |
| --- | --- |
| Navn | Studienummer |
| Fatima Kodro | 201609565 |
| Søren Bech | 201604784 |

**Jenkins build jobs**

Unit og coverage:

<http://ci3.ase.au.dk:8080/job/Team18103HandIn3_UnitAndCoverage/>

Integration:

<http://ci3.ase.au.dk:8080/job/Team18103HandIn3_Integration/>

Software metrics:

<http://ci3.ase.au.dk:8080/job/Team18103HandIn3_SoftwareMetrics/>

**Github repository**

<https://github.com/FatimaAU/HandIn3>

Indhold

# 

# Indledning

Der skal i aflevering 3, laves et system der kan overvåge luftfarten i et givent område, samt kunne holde øje med interessante event der måtte forekomme.

Problemet med luftfarten, er at der er mange fly, der flyver omkring samme område. Dette kan medføre stor farer, hvis nogle fly kommer for tæt på hinanden. Derfor skal hvert fly kunne registreres med position, fart og højde, for at sikre sig at ingen fly er for tætte på hinanden.

# Krav

Systemet skal kunne overvåge et bestemt luftrum, og registrere alle fly inden for dette luftrum, og ingen andre. Alle fly registreres med Tag, position, højde, fart(m/s) og kurs(grader).

Hvis nogle fly skulle flyve for tæt på hinanden (5000 meter horisontalt og 300 meter vertikalt) skal der kunne hæves et event indtil de er ude for farer igen. Når dette event er hævet, skal de fly der er for tætte på hinanden udskrives til en fil, med både flyenes tag og tidsrummet.

Der blev udgivet en Transponder Receiver dll i starten af afleveringen, som kunne modtage transponder data for flyene i luftrummet. Ud fra disse data, kunne flyenes Tag, position, højde registreres, samt tidspunktet for de forskellige transponder data.

# Design

Generel software design, hvordan har vi tænkt os at implementere og teste det. Hvordan vil vi dele opgaverne op. Hvordan har vi brugt CI serveren. Gode/dårlige ting skal forklares.

Plan over implementering

I starten af afleveringen blev der udgivet en dll-fil, hvorefter der blev arbejdet sammen i gruppen med at få printet de simple strenge ud. Herefter når dette var muligt, blev der oprettet nogle klasser, som kunne håndtere dette data. Det, som skulle håndteres, var at dataen blev parsed korrekt, samt at tidsstemplet blev omdannet og printet ud på en bestemt måde. Der blev herefter lavet test til disse klasser.

Herefter startede den egentlige implementeringsfase, som inkluderer resten af systemet. Der blev her anvendt TDD i en blanding af pair-programming og individuel programmering, hvor opgaverne blev delt ud. Dette gav mening idét systemet har mange dele, og det ville være en gode idé at definere krav (og test) før koden blev implementeret, dvs. der blev defineret en feature list.

Hvis en fejl fandtes i implementeringen, ville en test blive lavet omkring fejlen, og derefter ville den blive rettet.

Herefter blev koden refaktoreret ind i klasser. Koden havde en naturlig lav kobling, da tests kræver at en klasse eller kode skal være afkoblet fra sine afhængigheder.

Efter koden nu var blevet refaktoreret ind i klasser, skulle der laves unit tests til alle klasser. Dette blev opdelt i grupperne, så begge gruppemedlemmer havde ansvar for test af forskellige klasser.

## Klassediagrammer

## Sekvensdiagrammer

## Dependency tree

Hvilken strategi har vi valgt til integrationstesten. Billede af træet og forklaringer. Integration plan. **Se nederst i dokumentet for skabelon for sidste gang**.

# Klasser

Kort beskrivelse af hvad klasserne gør og måske udsnit af noget?

# Unit tests

# Integration tests

# Software metrics

# Konklusion

Hvordan har vi brugt CI serveren. Gode/dårlige ting skal forklares.

# Jenkins

Integration kører efter unit testene er stabile. Det giver ikke mening at lade det køre hvis denne test ikke er stabil.

SoftwareMetrics kører efter integration er færdig.

Coverage på unit testen. Integration skal ikke teste coverage af interfacene!

Upstream projekter: Før man kommer til denne test skal upstream projekter køre først.

Lavet pipeline view: Viser pipeline view dvs. hvilke jobs der kører når et andet er færdigt og hvilke der er stabile/ustabile eller aldrig kørt.

Behøver ikke bygge hver gang: lave et job som blot bygger hvormed de andre jobs som coverage kan tage de byggede filer og tjekke herpå. Dette er ikke gjort ..

kunne ikke finde den med "copy from artefacts" ...

På Tabel 1 ses en tabel over hvilke klasser der blev brugt i de forskellige trin.

D: Dette modul er inkluderet, og den der er drevet   
X: Dette modul er inkluderet  
S: Dette modul er stubbed eller mocked

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Step # | Output | Display | Power-Tube | Light | Cook-Controller | Timer | User-Interface | Button | Door |
| 1 | S | D | S |  | X | S | S |  |  |
| 2 | S | S | D |  | X | S | S |  |  |
| 3 | S | S |  | D | S |  | X | S | S |
| 4 |  | S | S |  | X | D |  |  |  |
| 5 | S | X | X | S | D | S | S | S | S |
| 6 | S | X | X | X | X | S | D | S | S |
| 7 | S | X | X | X | X | X | X | D |  |
| 8 | S | X | X | X | X | X | X |  | D |