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

Kort forklaring af formålet med opgaven og hvad rapporten indeholder. /fra Introduktion og 1.1

# Krav

Forklare kort de givne krav i opgaven. /fra punkt 2 og lidt om Transponder Receiver dll, hvad den gør og hvordan den bruges. Transponder data format /fra punkt 4.

# 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

Startede med at prøve at få de simple strenge ud fra den givne dll-filen, som der blev gjort sammen i gruppen. Herefter når dette var muligt, blev der oprettet klasser som kunne håndtere dataen, dvs. at den blev parsed korrekt, tidsstemplet blev omdannet og blev printet ud. Test blev herefter lavet. Herefter startede den egentlige implementeringsfase som inkluderer resten af systemet. Planen er at der skal anvendes TDD i en blanding af pair-programming og individuel programmering hvor opgaverne deles ud. Dette giver mening idét systemet har mange dele og det vil være en god idé at definere krav (og test) før koden implementeres. Dvs. en feature list er planlagt at blive defineret. Hvis en fejl findes, vil en test blive lavet for at se at fejlen findes, hvorefter fejlen rettes. Herefter vil koden refaktoreres ind i klasser. Koden vil desuden have en naturlig lav kobling, da tests kræver at en klasse eller kode skal være afkoblet fra sine afhængiger.

## 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 |