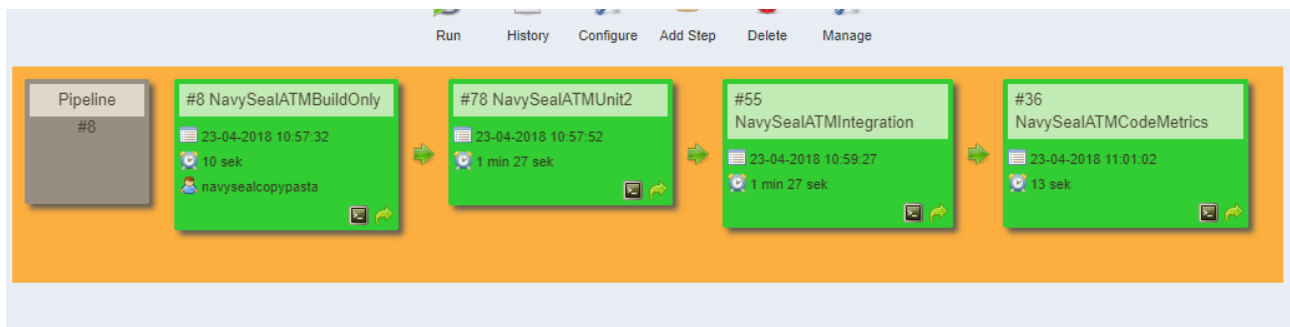


Gruppe 13

Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)

I4SWT Handin 3 – Air Traffic Monitor (ATM)

Gruppenummer: 13**Gruppemedlemmer:** Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)**Jenkins Unit Test:**<http://ci3.ase.au.dk:8080/job/NavySealATMUnit2/>**Jenkins Integrationstest:**<http://ci3.ase.au.dk:8080/job/NavySealATMIntegration/>**Jenkins Software Metrics:**<http://ci3.ase.au.dk:8080/job/NavySealATMCodeMetrics/>**Jenkins Pipeline link:**<http://ci3.ase.au.dk:8080/user/navysealcopyasta/my-views/view/NavySealATM/>

Figur 1 - Jenkins pipeline

1. Introduktion

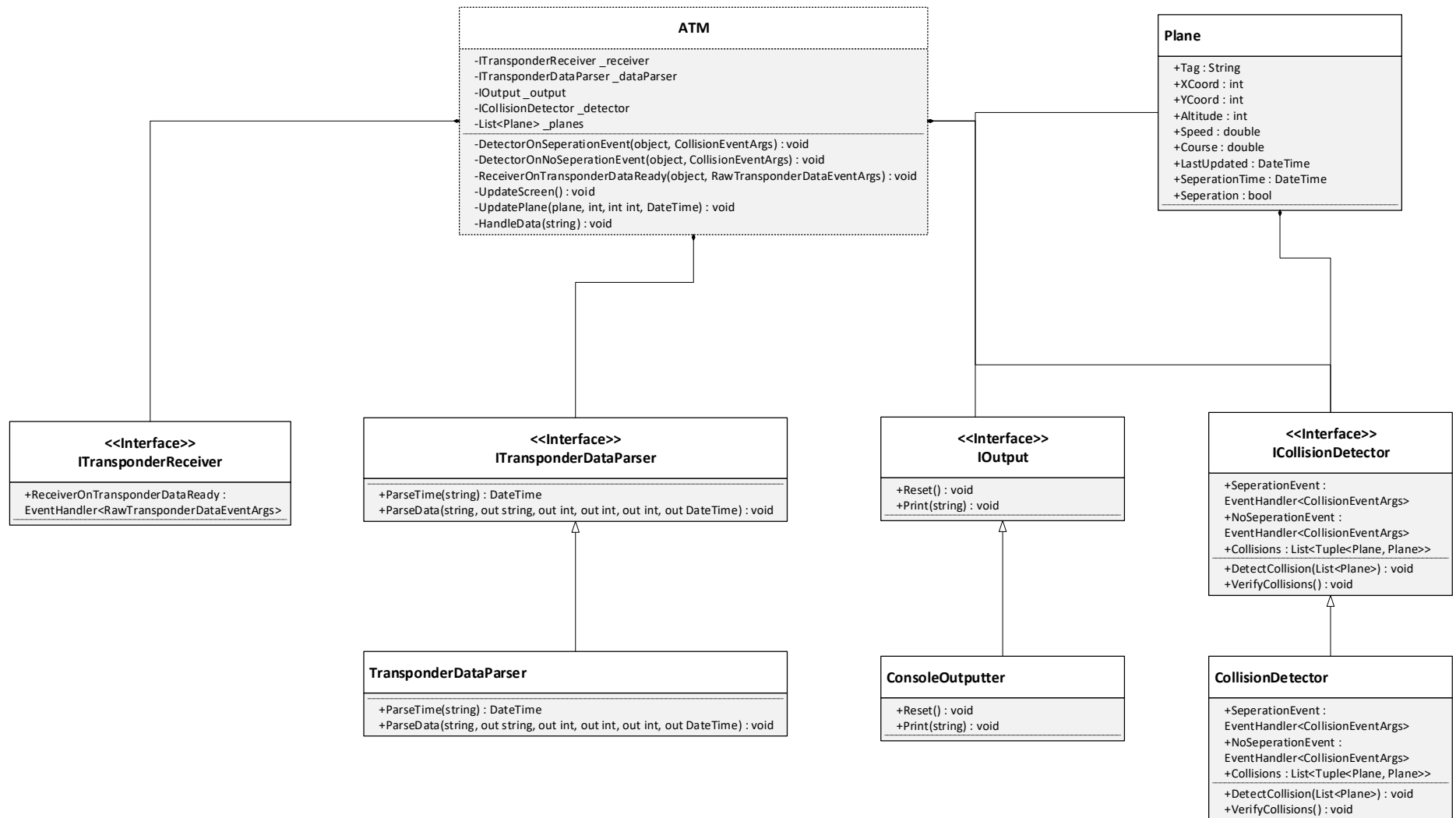
Denne øvelse går ud på at lave et ATM-system som kan overvåge et bestemt område og præsentere den flytrafik som befinder sig heri. Ydermere skal systemet kunne detektere interessante events i trafikken, såsom hvis to eller flere fly kommer for tæt på hinanden.



Figur 2 - Flytrafik over verden. Gul og orange: Kommerciel flyvning

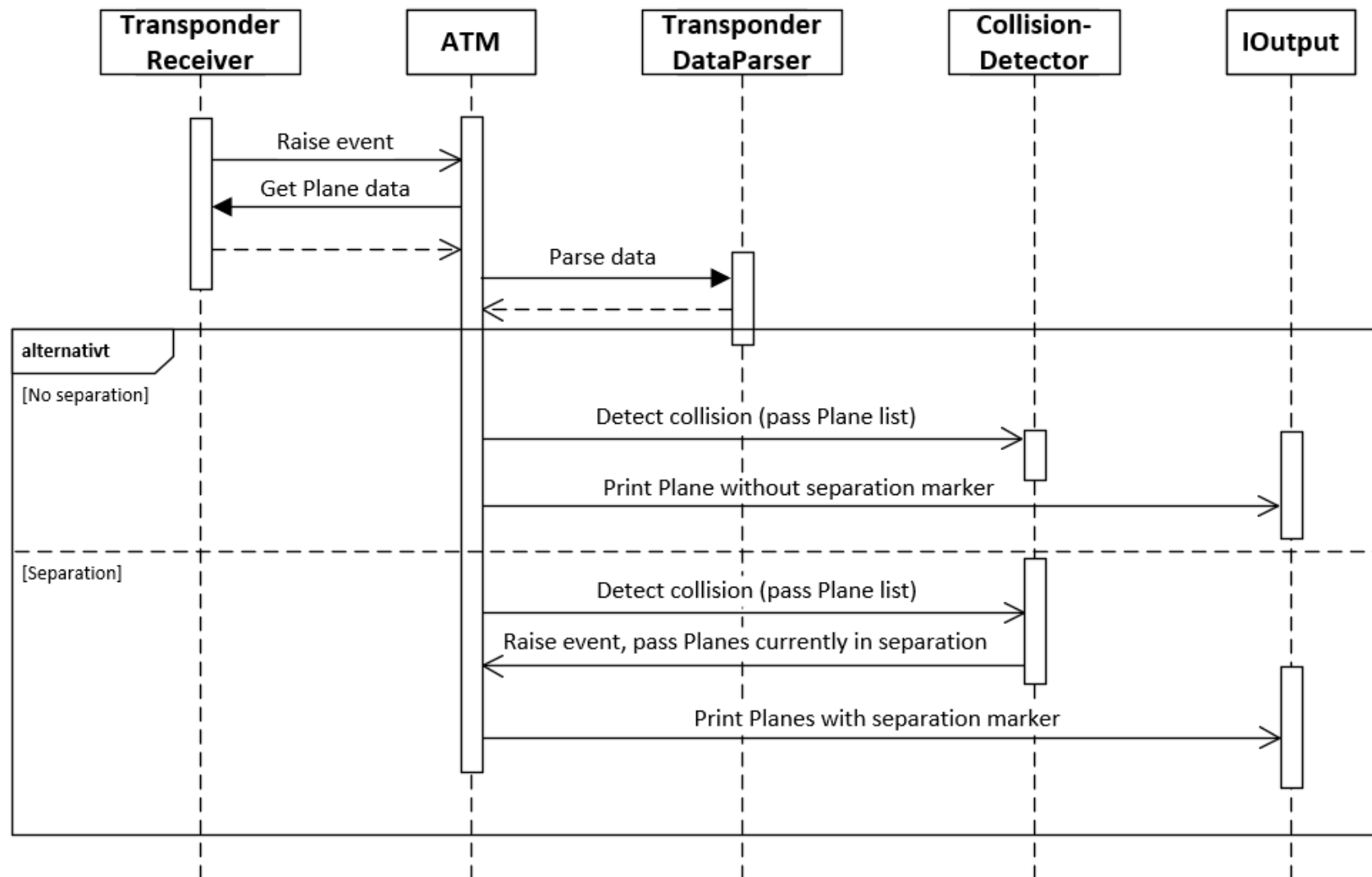
Alle fly har en *transponder* som udsender information vedrørende flyet. Alle data opfanges og kan bruges i fx en lufthavn, til at få information om alle fly og vigtige events vedrørende disse.

2. Klassestruktur



Figur 3 - Klassediagram for Air Traffic Monitor

2.1. Sekvensdiagram



Figur 4 - Sekvensdiagram

Gruppe 13

Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)

2.1. Beskrivelse af klasser

ATM: Er ansvarlig for at koble alle klasserne sammen. ATM er den primære klasse som bliver tilføjet til konsolapplikationen. *ATM* kobler ved hjælp af en eventhandler op til *ITransponderReceiver* og sørger for at sende denne data videre til *ITransponderDataParser*. Når dataen er blevet parset bliver flyene opbevaret i *ATM* klassen, som også håndterer separation events. *ATM* klassen kan dermed også ses som et alternativ til en større konsol/Main program.

ITransponderReceiver: Er det library som genererer flyoplysninger. Denne er udleveret af lektoren.

ITransponderDataParser: Er ansvarlig for at processere det data, som bliver modtaget fra *ITransponderReceiver*. Denne klasse modtager en semikolon-separeret *string*, som derefter bliver splittet op i X/Y-koordinater, hastighed, retning og tidsstempel.

IOutput: Skriver til konsollen. Her er benyttet et interface med en simpel *Print()* funktion for at have mulighed for f.eks. at printe til en fil eller en database i fremtiden.

ICollisionDetector: Modtager et array med *Plane* og finder ud af om disse er på kollisionskurs, hvis inden for en bestemt afstand. I tilfælde af at to fly er ved at kolliderer, raiser den et event. Klassen indeholder også en metode kaldet *VerifyCollision()* som raiser et event i tilfælde af at der er to fly, der ikke længere kolliderer med hinanden.

Plane: Er en klasse som indeholder data for et fly i form af public properties

2.2. Fordeling af arbejde

I forhold til fordeling af klasserne, unit tests og integrationstests, har gruppen arbejdet således:

Simon: *CollisionDetector* (inkl. UT) og *IT1*

Jonas: *ATM* og *DataParser*

Søren: UT for *DataParser*, *Plane* og *ATM*. *IT2*

Magnus: UT for *ATM* og *IT3*

Simon og Jonas har hovedsageligt arbejdet på selve klasserne, mens Søren og Magnus har arbejdet med Unit tests og integrationstests.

Årsagen til denne fordeling af arbejdet har været for at muliggøre sideløbende arbejde med tests og implementation. Her kan nævnes en form for udvikling der minder om test-driven development, TDD, hvorved de enkelte tests til softwaren er skrevet umiddelbart samtidigt som udviklingen. Dette har medført at der har været konstant feedback til udviklingen, selv uden at testene er blevet skrevet først.

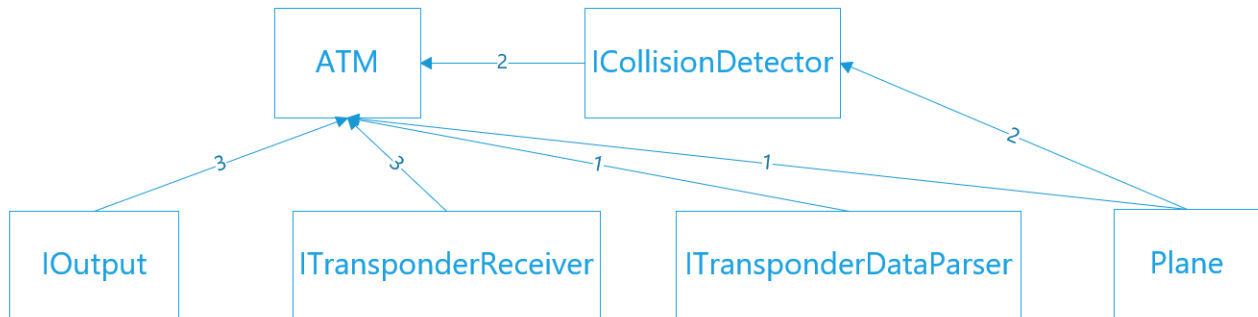
3. Unit Test

Der er blevet lavet unit test af de 4 forskellige klasser: *ATM*, *CollisionDetector*, *Plane* og *TransponderDataParser*. Målet med disse Unit tests er for at sikre, at klasserne udfører de rigtige operationer bl.a. ved hjælp af Boundary Value Analysis og mocking af dependencies.

Til test af *ConsoleOutputter* er foretaget manuelle tests, som tjekker om hvorvidt det virker. Til kontrol er taget billeder af succesfulde tests.

4. Integration

I forhold til integrationstest blev der først tegnet et dependency tree af klasserne. Dette dependency tree kan ses nedenfor.



Figur 5 - Dependency tree

Det kan ses at *ATM* er hovedklassen, som forbinder de øvrige klasser. Derfor ville det give mening at tage udgangspunkt fra denne, når der skal laves integrationstest med udgangspunkt fra *ATM* som driver.

Plane er også tilføjet som klasse, men da *Plane* kun indeholder properties, har gruppen besluttet, at lade denne klasse blive testet under Unit Test og ikke integrationstest. Fokus på integrationstest er dermed at teste forholdet mellem *ATM* og integrationen med de øvrige klasser.

#Step	ATM	Plane	Transp..DataParser	CollisionDetector	ConsoleOutputter	Transp..Receiver
1	D	X	X	S	S	S
2	D	X	X	X	S	S
3	X	X	X	X	X	X

Tabel 1 – Integrationsplan

S: Modulet er substitueret eller mocket

D: Modulet er inkluderet og den som er drevet under test

X: Modul er inkluderet

Det er besluttet at bruge collaboration integration, da både *ATM* og *CollisionDetector* klasserne er afhængige af *Plane* klassen. Der er også flere særskilte moduler som kalder forskellig funktionalitet i *ATM*, det var derfor mere fordelagtigt at splitte integrationen op i de integrationer som *ATM* havde med de øvrige klasser.

For første integrationstrin er *ATM* driver og *Plane* samt *TransponderDataParser* testet. Målet ved dette integrationsstep er at vise, at fly bliver tilføjet i korrekte *Plane* klasser, når der bliver sendt information fra en stubbet *TransponderReceiver*. Selve udførelsen sker via et event som bliver raised gennem *NSubstitute*. Der er benyttet klassen *RawTransponderEventArgs* som event arguments til *ATM*.

Andet integrationstrin er at tilføje *CollisionDetector* til *ATM*. Her bliver der testet om de korrekte events bliver raised i tilfælde af at kolliderende fly bliver tilføjet til *ATM*. Disse events skal håndteres korrekt og det skal bevises at *ATM* kan håndtere separation events.

Det sidste integrationstrin involverer en konsolapplikation, som både tester *ConsoleOutputter* og *TransponderReceiver*. Disse er testet til sidst sammen med *ATM*, da *TransponderReceiver* giver random

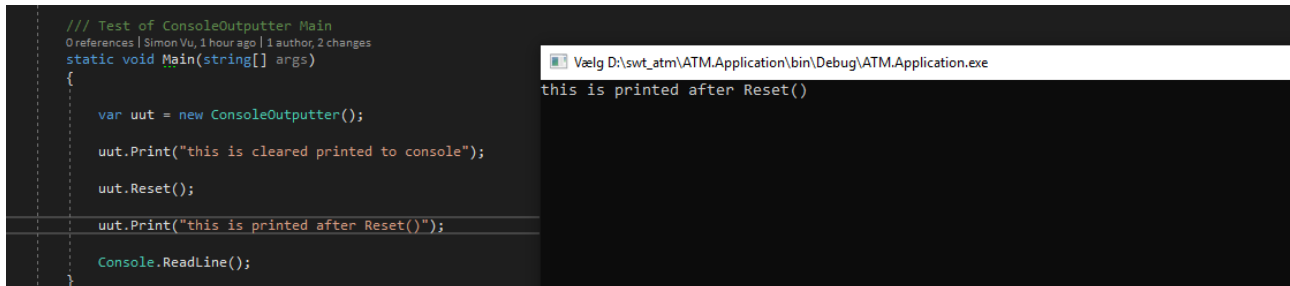
Gruppe 13

Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)

værdier som gør den svær at teste uden manuel test på konsollen. *ConsoleOutputter* er i større grad blevet testet under unit test, så denne kan godt placeres til sidst sammen med *TransponderReceiver*.

5. Resultater

5.1. Test af IOutput



```
/// Test of ConsoleOutputter Main
References | Simon Vu, 1 hour ago | 1 author, 2 changes
static void Main(string[] args)
{
    var uut = new ConsoleOutputter();

    uut.Print("this is cleared printed to console");

    uut.Reset();

    uut.Print("this is printed after Reset()");

    Console.ReadLine();
}
```

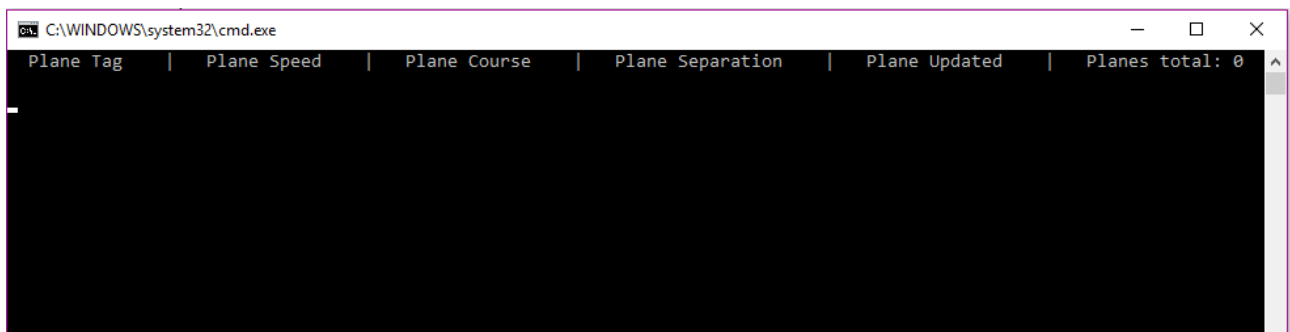
Vælg D:\swt_atm\ATM.Application\bin\Debug\ATM.Application.exe

this is printed after Reset()

Figur 6 - Test af Reset()

5.2. Endelig integration

I dette afsnit kan der findes screenshots af det endelige konsolprogram med alle klasserne integreret. Nedenfor ses konsolvinduet med *ATM* klassen uden nogle fly fra *TransponderReceiver*.



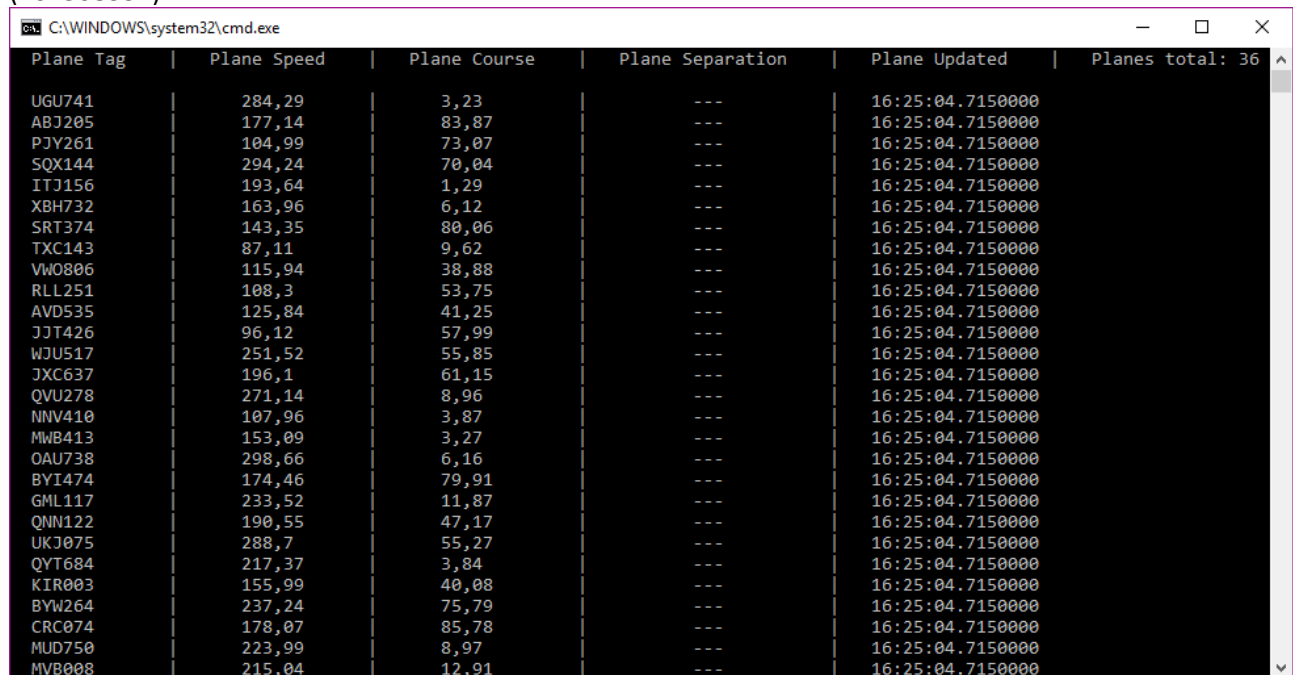
```
C:\WINDOWS\system32\cmd.exe
```

Plane Tag	Plane Speed	Plane Course	Plane Separation	Plane Updated	Planes total: 0

Figur 7 - Console output uden fly

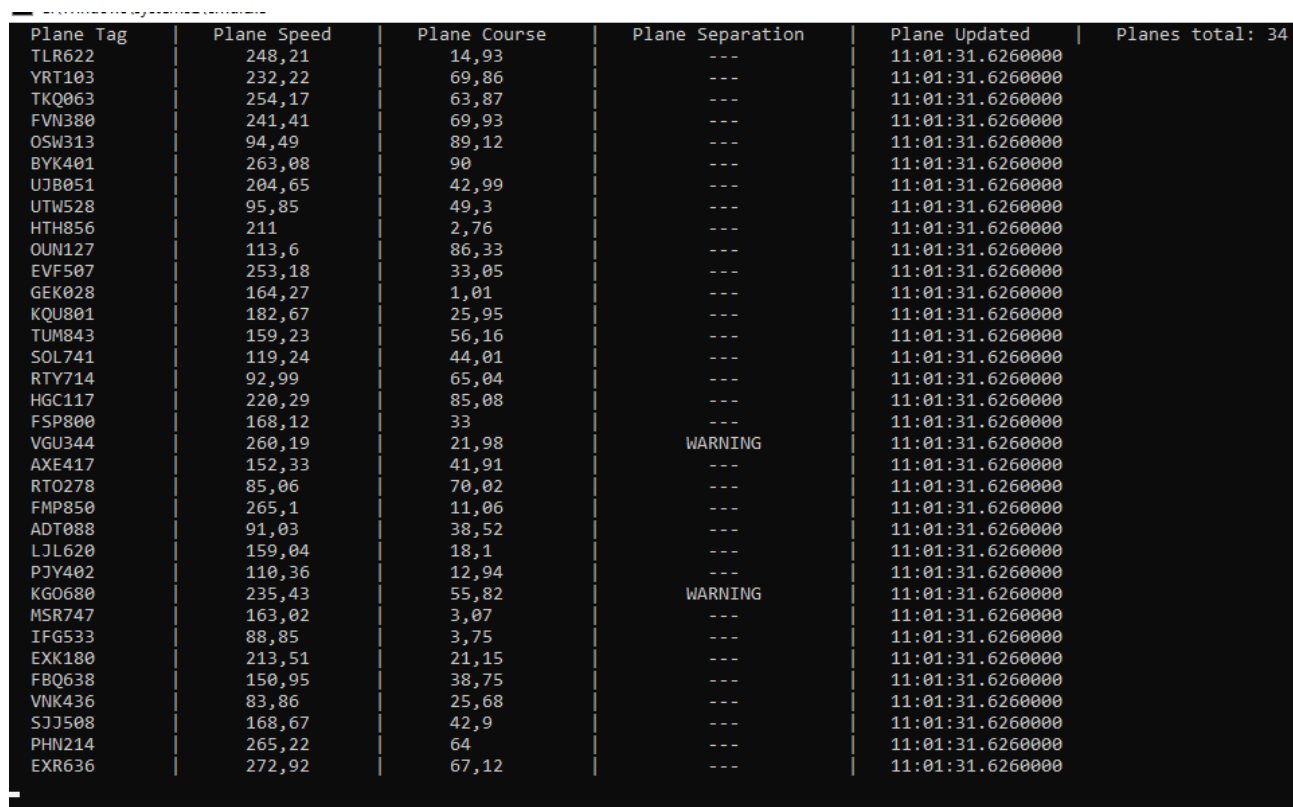
Gruppe 13

Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)



Plane Tag	Plane Speed	Plane Course	Plane Separation	Plane Updated	Planes total: 36
UGU741	284,29	3,23	---	16:25:04.7150000	
ABJ205	177,14	83,87	---	16:25:04.7150000	
PJY261	104,99	73,07	---	16:25:04.7150000	
SOX144	294,24	70,04	---	16:25:04.7150000	
ITJ156	193,64	1,29	---	16:25:04.7150000	
XBH732	163,96	6,12	---	16:25:04.7150000	
SRT374	143,35	80,06	---	16:25:04.7150000	
TXC143	87,11	9,62	---	16:25:04.7150000	
VW0806	115,94	38,88	---	16:25:04.7150000	
RLL251	108,3	53,75	---	16:25:04.7150000	
AVD535	125,84	41,25	---	16:25:04.7150000	
JJT426	96,12	57,99	---	16:25:04.7150000	
WJU517	251,52	55,85	---	16:25:04.7150000	
JXC637	196,1	61,15	---	16:25:04.7150000	
QVU278	271,14	8,96	---	16:25:04.7150000	
NNV410	107,96	3,87	---	16:25:04.7150000	
MWB413	153,09	3,27	---	16:25:04.7150000	
OAU738	298,66	6,16	---	16:25:04.7150000	
BYI474	174,46	79,91	---	16:25:04.7150000	
GML117	233,52	11,87	---	16:25:04.7150000	
QNN122	190,55	47,17	---	16:25:04.7150000	
UKJ075	288,7	55,27	---	16:25:04.7150000	
QYT684	217,37	3,84	---	16:25:04.7150000	
KIR003	155,99	40,08	---	16:25:04.7150000	
BYW264	237,24	75,79	---	16:25:04.7150000	
CRC074	178,07	85,78	---	16:25:04.7150000	
MUD750	223,99	8,97	---	16:25:04.7150000	
MVB008	215,04	12,91	---	16:25:04.7150000	

Figur 8 - Console output med fly



Plane Tag	Plane Speed	Plane Course	Plane Separation	Plane Updated	Planes total: 34
TLR622	248,21	14,93	---	11:01:31.6260000	
YRT103	232,22	69,86	---	11:01:31.6260000	
TKQ063	254,17	63,87	---	11:01:31.6260000	
FDN380	241,41	69,93	---	11:01:31.6260000	
OSW313	94,49	89,12	---	11:01:31.6260000	
BYK401	263,08	90	---	11:01:31.6260000	
UJB051	204,65	42,99	---	11:01:31.6260000	
UTW528	95,85	49,3	---	11:01:31.6260000	
HTH856	211	2,76	---	11:01:31.6260000	
OUN127	113,6	86,33	---	11:01:31.6260000	
EVF507	253,18	33,05	---	11:01:31.6260000	
GEK028	164,27	1,01	---	11:01:31.6260000	
KQU801	182,67	25,95	---	11:01:31.6260000	
TUM843	159,23	56,16	---	11:01:31.6260000	
SOL741	119,24	44,01	---	11:01:31.6260000	
RTY714	92,99	65,04	---	11:01:31.6260000	
HGC117	220,29	85,08	---	11:01:31.6260000	
FSP800	168,12	33	---	11:01:31.6260000	
VGU344	260,19	21,98	WARNING	11:01:31.6260000	
AXE417	152,33	41,91	---	11:01:31.6260000	
RT0278	85,06	70,02	---	11:01:31.6260000	
FMP850	265,1	11,06	---	11:01:31.6260000	
ADT088	91,03	38,52	---	11:01:31.6260000	
LJL620	159,04	18,1	---	11:01:31.6260000	
PJY402	110,36	12,94	---	11:01:31.6260000	
KGO680	235,43	55,82	WARNING	11:01:31.6260000	
MSR747	163,02	3,07	---	11:01:31.6260000	
IFG533	88,85	3,75	---	11:01:31.6260000	
EXK180	213,51	21,15	---	11:01:31.6260000	
FBQ638	150,95	38,75	---	11:01:31.6260000	
VNK436	83,86	25,68	---	11:01:31.6260000	
SJJ508	168,67	42,9	---	11:01:31.6260000	
PHN214	265,22	64	---	11:01:31.6260000	
EXR636	272,92	67,12	---	11:01:31.6260000	

Figur 9 - Console output med separation event

Ovenfor ses screenshots af konsolapplikationen, både med separation event og uden separation event. Disse screenshots beviser at integrationen mellem de forskellige klasser er udført korrekt.

Gruppe 13

Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)

6. Software metrics

Software metrics er også blevet udført i Jenkins pipelinen. Software metrics bliver automatisk udført efter integrationstest på Jenkins serveren. Et screenshot af de målte software metrics for projektet kan ses nedenfor.

Namespace Summary

Name	MaintainabilityIndex (dff)	CyclomaticComplexity (dff)	ClassCoupling (dff)	DepthOfInheritance (dff)	LinesOfCode (d
TransponderLib	87	85	24	2	153

Code Metrics by Type

Name	MaintainabilityIndex (dff)	CyclomaticComplexity (dff)	ClassCoupling (dff)	DepthOfInheritance (dff)	LinesOfCode (d
ICollisionDetector	100	6	4	0	0
TransponderDataParser	68	4	3	1	17
CollisionEventArgs	93	5	2	2	7
Plane	92	20	1	1	20
IOutput	100	2	0	0	0
ITransponderDataParser	100	2	1	0	0
ATM	61	19	17	1	68
CollisionDetector	71	24	10	1	38
ConsoleOutputter	97	3	2	1	3

Figur 10 - Software metrics

Det generelle maintainability index er umiddelbart godt. Der er nogle få klasser som f.eks. *ATM*, *TransponderDataParser* og *CollisionDetector*, som har et mindre godt maintainability index.

Cyclomatic Complexity for metoderne overstiger ikke 10. Der kan argumenteres for at nogle funktioner såsom *CollisionDetector.DetectCollision()* burde blive splittet op i to metoder, da de har 10 i cyclomatic complexity. Dette er der dog besluttet for ikke at gøre noget ved.

I forhold til coupling mellem klasserne har *ATM* klassen betydeligt større kobling mellem de øvrige klasser. Dette giver god mening, da *ATM* er klassen som binder de andre klasser.

7. Continuous integration med Jenkins

Til CI Jenkins benyttet for at oprette en pipeline til automatiske Unit tests, integrationstests og softwaremetrics. Ved at koble gruppen Github repository op til Jenkins, har gruppen løbende fået feedback i form af fejlede/succesfulde tests samtidig med kodning. Gruppen har i høj grad forsøgt at *pushe* til det delte repository så ofte som muligt for at få hurtig feedback.

Pipelinen har især været brugbar i udarbejdelsen af integrationstest. Når integrationstests skulle skrives, var der nogle få situationer, hvor det blev nødvendigt at ændre lidt i klasserne. Her har CI hjulpet meget, da små ændringer i forbindelse med integrationstest hurtigt blev reflekteret i unit tests, i tilfælde af at ændringerne havde effekt på resten af systemets funktionalitet.

Det hjalp også meget at hele processen er automatiseret, så det ikke er nødvendigt manuelt at sætte pipelinen i gang. Dermed opnås mere data for eventuelle fejl.

Gruppe 13

Simon Vu (201606029), Magnus Dewett (201511612), Søren Skieller (201508090) og Jonas Agger (201508334)

8. Konklusion

Det er lykkedes at lave en ATM som overvåger et bestemt område givet i øvelsen og efterfølgende præsenterer de data der er modtaget fra transponderReceiveren. Når der er mulighed for kollision mellem to, eller flere, fly, skrives en advarsel til skærmen.