LADESKAB

Obligatorisk aflevering 2

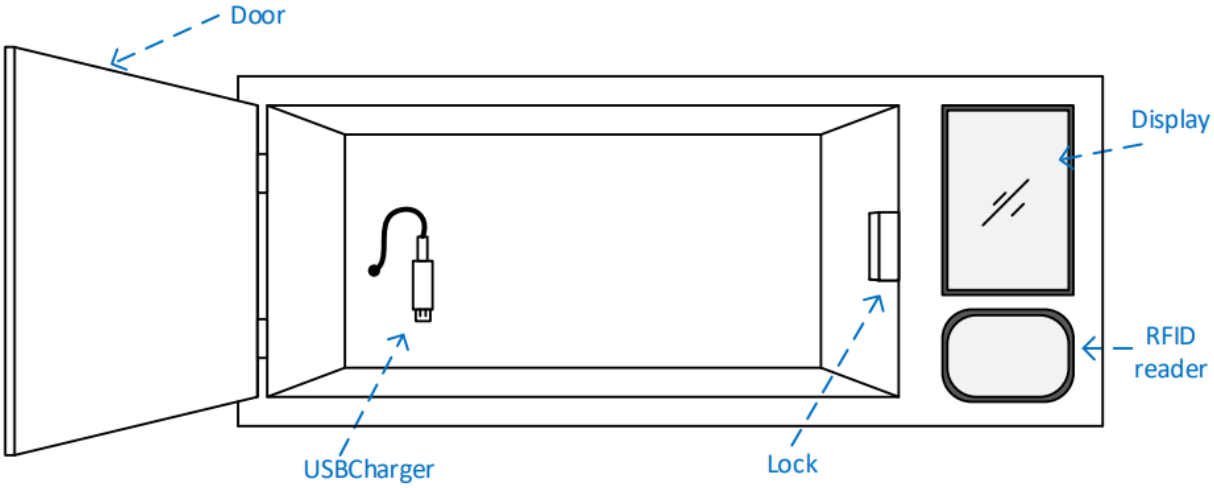

Figure 1: Principskitse af ladeskabet

Casper Fevre Hansen 201501949
Michael Juhl 20084844
Jim Sørensen 201602614

Gruppe: 34

Date: 2021-03-04

Contents

1 Indledning

Denne rapport beskriver den 2. obligatoriske handin i faget Softwaretest. Rapporten er blevet udarbejdet i fællesskab af gruppe 34.

Selve rapporten omhandler design, implementering og test af software til et ladeskab til en mobiltelefon, som skal opstilles i omklædningsrummet i en svømmehal eller lignende. Formålet med ladeskabet er, at en bruger kan tilslutte sin mobiltelefon til opladning i ladeskabet, efterlade sin telefon og senere afhente den igen. Som identifikation tænkes en RFID tag der hænger på badebåndet eller nøglen til et tøjskab.

Link til GitHub-repositoriet:
https://github.com/TeamSWT34/Grup34_Ladeskab

Link til Jenkins-jobbet:
http://ci3.ase.au.dk:8080/job/Gruppe34_Ladeskab/

2 Dokumentation

2.1 Ladeskabssystem

Ladeskabet bliver brugt som følger (det antages at skabet ikke er i brug fra start): Dette bliver simulerer i et konsol-vindue som kan ses på figur 2.

Konsole er dele op i tre dele, "Station msg" viser visulade ting så som, Dørren er åben eller lukket. "Charge msg" Viser beskeder fra oplader, fejl bedskeder hvis mobil er tilslutte forkert og hvor mange strøm der er på den.

- Brugeren åbner lågen på ladeskabet
  - Dette simuleres ved at der bliver trykket o-O for "Open" i konsollen.
- Brugeren tilkobler sin mobiltelefon til ladekablet.
  - Programmet antager at mobilen er sat i fra start - Der mulighed for at fjerne den ved at trykkes på u-U for "Usb" dette simulere at en mobil afbrudt fra system, "trykke der igen vil den blive tilslutte igen".
- Brugeren lukker lågen på ladeskabet.
  - Dette simuleres ved at der bliver trykket c-C for "Close".
- Brugeren holder sit RFID tag op til systemets RFID-læser.
  - Der trykkes på r-R for "RfReaed".
- Systemet aflæser RFID-tagget. Hvis ladekablet er forbundet, låser systemet lågen på ladeskabet, og låsningen logges. Skabet er nu optaget.
  - Dette simulere ved at der trykkes en kode/RfId ind i konsollen, som kan ses på figur: 3.

Opladning påbegyndes. Brugeren kan nu forlade ladeskabet og komme tilbage senere. Imens vise displayet/konsollen en bruger besked og hvor mange strøm der er på mobilen dette kan ses på figur: 4. Herefter fortsætter den tiltænkte brug som følger:

- Brugeren kommer tilbage til ladeskabet.
- Brugeren holder sit RFID tag op til systemets RFID-læser.
  - Dette simulerer ved at trykke på r-R for "RfReaed".
- Systemet aflæser RFID-tagget. Hvis RFID er identisk med det, der blev brugt til at låse skabet med, stoppes opladning, ladeskabets låge låses op og oplåsningen logges dette kan ses på figur 5.
- Brugeren åbner ladeskabet, fjerner ladekablet fra sin telefon og tager telefonen ud af ladeskabet.
  - Dette simulerer ved at trykke på u-U for at fjerne mobilen.
- Brugeren lukker skabet. Skabet er nu ledigt.
  - Dette simulerer ved at trykke på c-C for lukke skabet igen.

Når der bliver låsning og oplåsning, bliver der logges en fil med timestamp (timer, minutter og sekunder), det aktuelle RFID samt en beskrivende tekst for begivenheden.

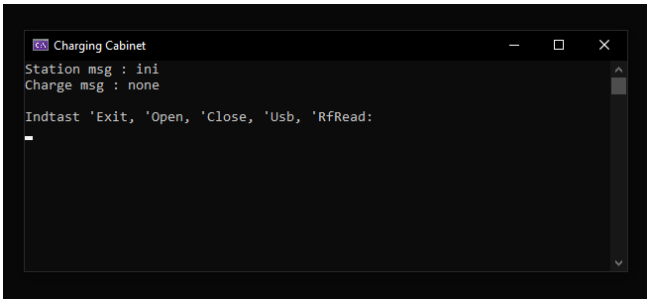

Figure 2: Init Console

Figure 3: RfId Input

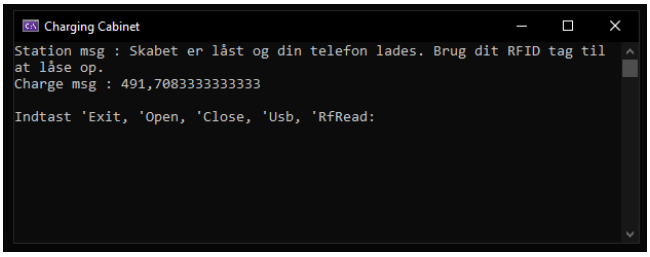

Figure 4: Charge Startet

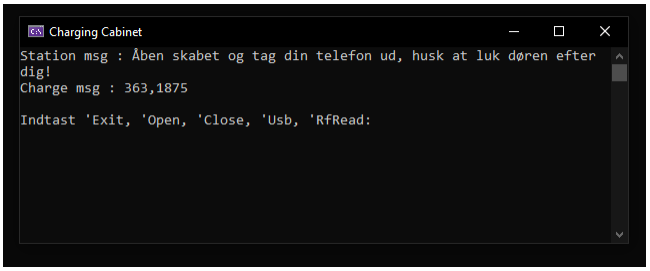

Figure 5: Correct RfId

2.2 Designe

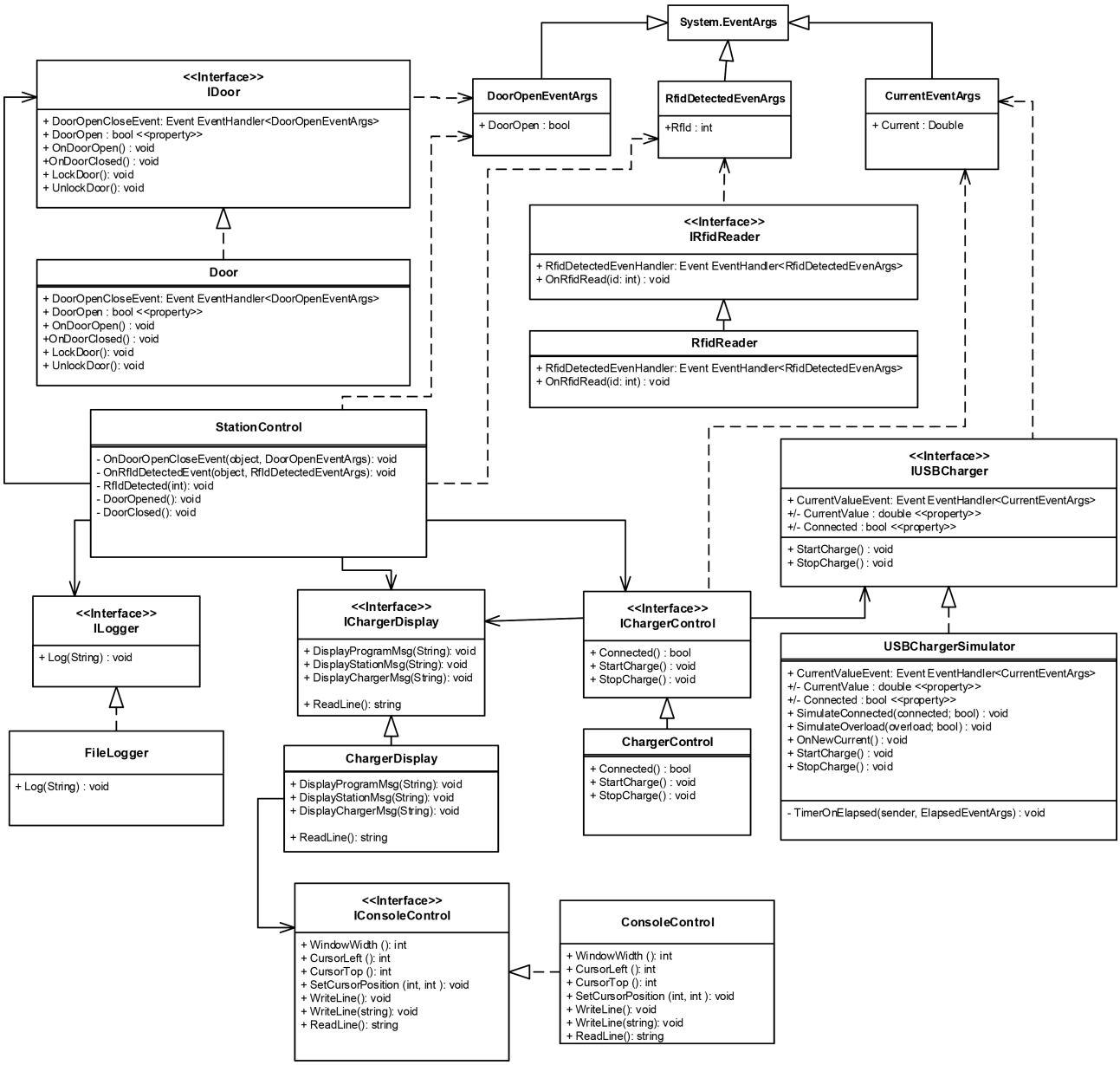

Figure 6: Class Diragram: All System

Bemærk at i alle sekens diagrammerne har lavet et kald fra ChargerControlDisplay til ConsoleControl hvor vi blot skriver "14 Calls". Dette har vi gjort for at spare plads.

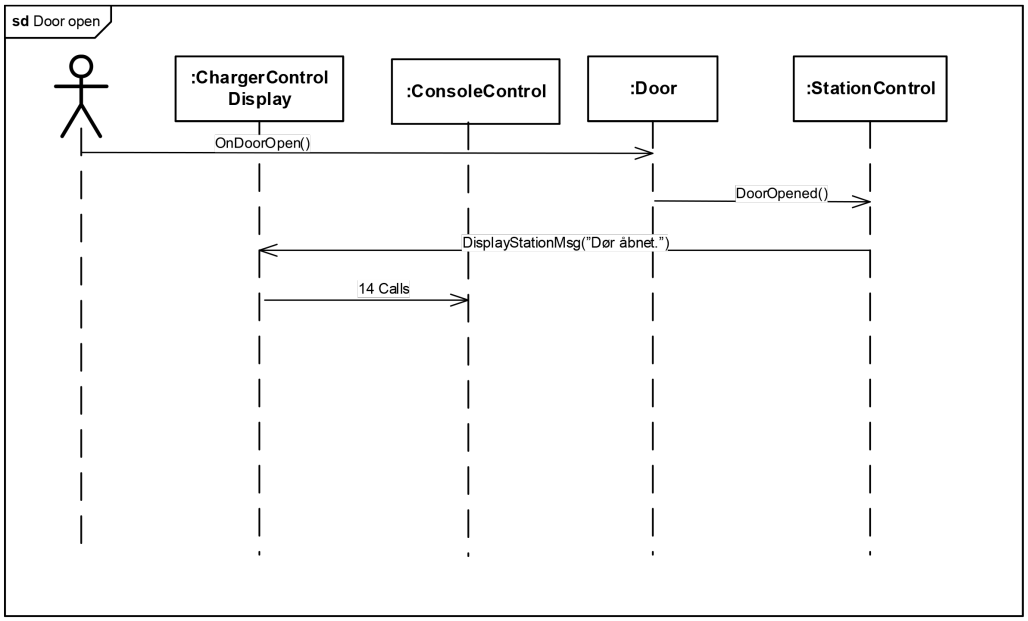

Figure 7: Sekvens Diragram: Open Door

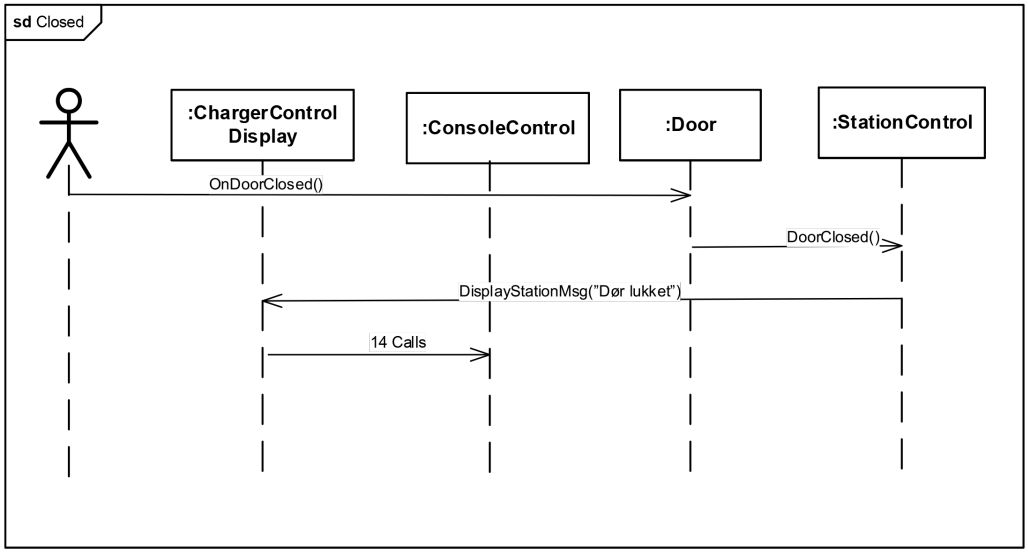

Figure 8: Sekvens Diragram: Close Door

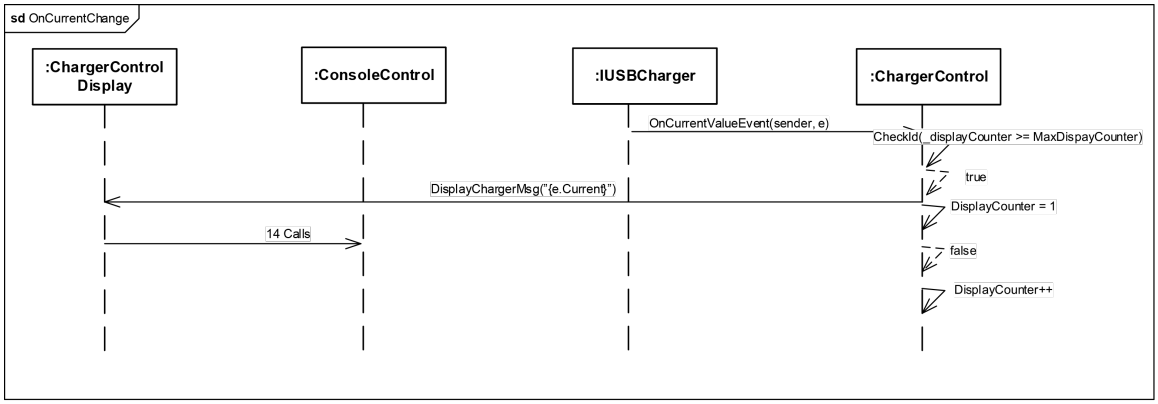

Figure 9: Sekvens Diragram: Current Change

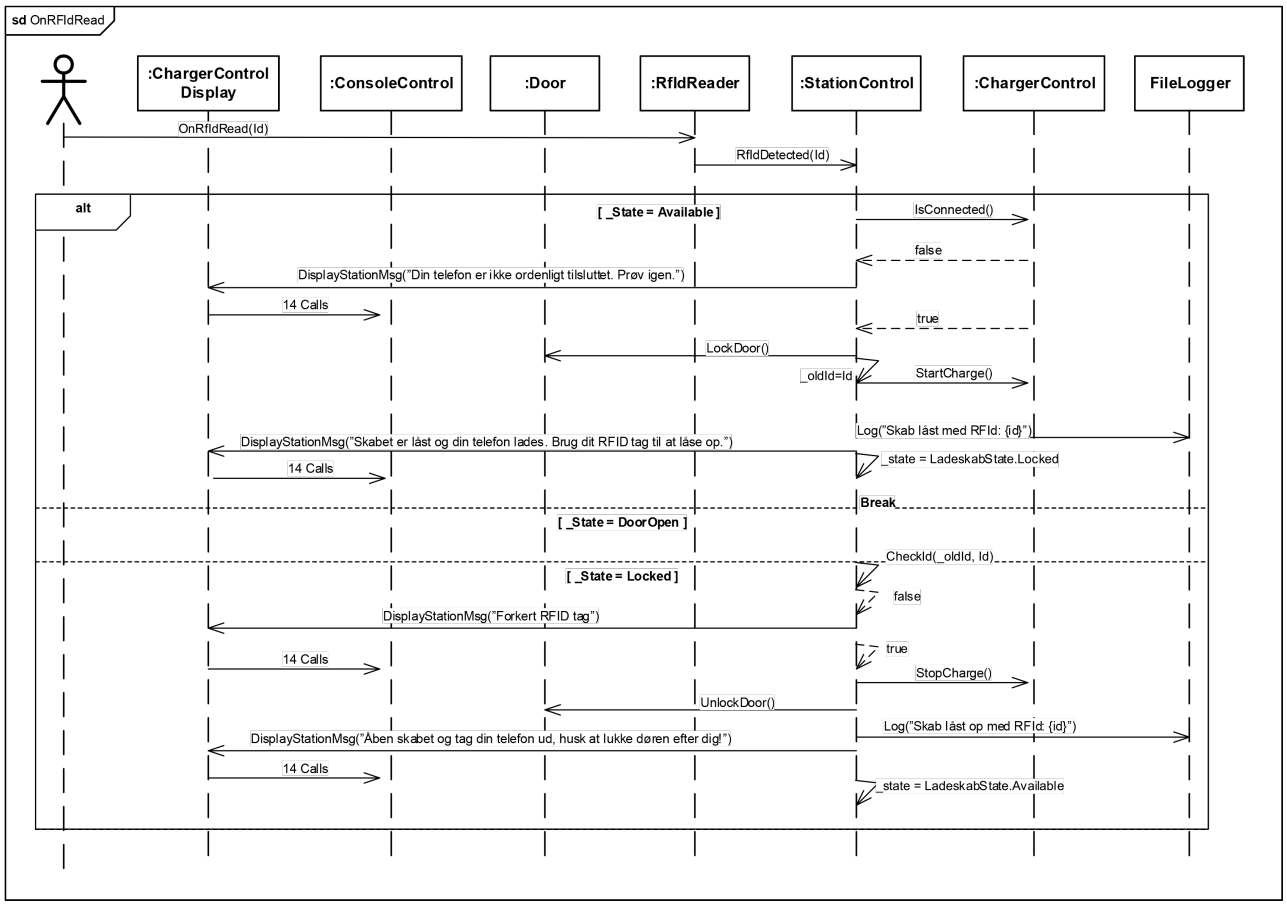

Figure 10: Sekvens Diragram: RfId Read

2.3 Implementere

Vi valgte at lave Interface for alle klasser undtagen StationControl dette kan ses i kode: 11. Dette gjorde vi for at gøre det lettere at lave stubs/mocks til unit testing.
IConsoleControl/ConsoleControl refactorede (SoR) vi ud af ChargerConsoleDisplay. Så det var lettere at teste dele der har "System.Console" dependency. Dette gave også fordelen af at teste kunne teste GUI-logic og console functioner for sig, interface til dette kan se på figur 2.1.

Listing 2.1: IConsoleControl

```
namespace ChargingCabinetLib.Interface
{
    public interface IConsoleControl
    {
        int WindowWidth { get; }
        int CursorLeft { get; }
        int CursorTop { get; }
        void SetCursorPosition(int left, int top);
        void WriteLine();
        void WriteLine(string text);
        string ReadLine();
    }
}
```

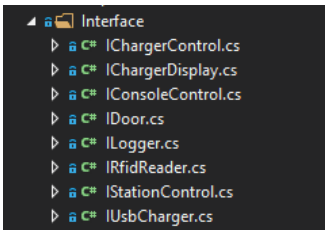

Figure 11: Interface-Code

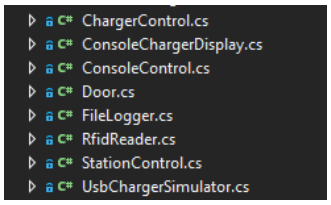

Figure 12: Class-Code

For at tage højde for det fysisk hardware, eksempel som display, har vi lavet et interface der kan bruges på hvilket som helt display, og ikke kun en konsol som vi har gjort, interface kan ses i kode: 2.2. Til konsol har vi så lavet endnu et interface der angiver de funktionalitetter som ChargerConsoleDisplay har behov for fra konsol.

Listing 2.2: IChargerDisplay

```
namespace ChargingCabinetLib.Interface
{
    public interface IChargerDisplay
    {
        void DisplayProgramMsg(string msg);
        void DisplayStationMsg(string msg);
        void DisplayChargerMsg(string msg);
        string ReadLine();
    }
}
```

Har for at visualisere de steder hvor der i koden skulle ske noget fysisk, så som at åbne døren til skabet eller at indsætte Rfid på læserne. Har vi benytte os af event. Eksempelvis ved at åben skabsdøren som kan ses i kode: 2.3.

Listing 2.3: OnDoorEvent

```
private void OnDoorEvent(bool eventArgValue) => DoorOpenCloseEvent?
                .Invoke(this, new DoorOpenEventArgs { DoorOpen = eventArgValue });

public event EventHandler<DoorOpenEventArgs> DoorOpenCloseEvent;
```

2.4 UnitTeste

Da vi havde interface til det hele havde vi mulighed for at bruge "stub/mock" dette gjorde det rimelig nemt for at at teste vores kode. I kode 2.4 kan ses setup for vores test af "ChargerControl", her benytter vi os af fake's. Dette gør vi for at kunne abstrahere afhængigheder ud.

Listing 2.4: Setup Example

```
1   private ChargerControl _uut;
2   private IUsbCharger _fakeUsbCharger;
3   private IChargerDisplay _fakeDisplay;
4   [SetUp]
5   public void Setup()
6   {
7       _fakeUsbCharger = Substitute.For<IUsbCharger>();
8       _fakeDisplay = Substitute.For<IChargerDisplay>();
9       _uut = new ChargerControl(_fakeUsbCharger, _fakeDisplay);
10  }
```

Et eksemple på "stub" kan ses i kode: 2.5 har vi en fake for IUsbCharger som vi Stub'er, dette gøres ved at sætte dens "Connected" retur værdig til "false".

Listing 2.5: Stub Example

```
1   [Test]
2   public void IsConnected_ReturnFalse()
3   {
4       _fakeUsbCharger.Connected.Returns(false);
5
6       Assert.IsFalse(_uut.IsConnected());
7   }
```

Eksemple på "mock" kan ses i kode: 2.6, dette for går ved at lave check, om "StartCharge UsbCharger Received()" har modtaget et kald på "StartCharge()".

Listing 2.6: Mock Example

```
1   [Test]
2   public void StartCharge_UsbCharger_Received()
3   {
4       _uut.StartCharge();
5       _fakeUsbCharger.Received().StartCharge();
6   }
```

At skulle teste på loggerne forvolde os dog lidt problemer, da vi skulle test på om der blev skrevet til log-filen. I kode: 2.7 fandt vi ud af ved at ved at bruge "Encoding.Unicode" fik vi et helt forkert resultat, vore test virker selvfølgelige ikke. Dette skyldes at vi fik omskrevet "res", den værdig vi sammeliger med til kinesisk. Vi fandt ud af dette ved at printe "res" værdigen ud i loggen som kan ses på figur 13. Vi fik løst dette ved at fjerne "Encoding.Unicode" men dette kunne også løse ved at "Encoding.UTF8" i stedet for.

Listing 2.7: $\text{Log}_{W}ritesToFileFAIL$

```
1       [TestCase("Test")]
2       [TestCase("")]
3       [TestCase(null)]
4       public void Log_WritesToFile(string text)
5       {
6           DateTime dt = DateTime.Now;
7           _uut.Log(text);
8
9           Assert.That(File.Exists(FileName));
10
11          string res;
12          using (StreamReader sR = new StreamReader(File.OpenRead(FileName), Encoding.Unicode))
13          {
14              res = sR.ReadLine();
15          }
16
17          _uut.Log(res);
18          Assert.That(res == $"{dt}: {text}");
19
20      }
```

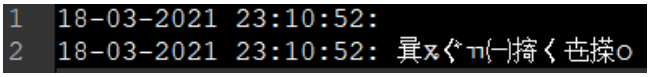

Figure 13: Log_WritesToFile Fail

Vores github koden kan finde på: 1[GitHub-Link] ellers kan ses i indledningen.

2.5 Jenkins

Helt i starten af arbejdet sat vi jenkin op. Før vi fik sat jenkin 'Web-Hook' op, lavede vi en del af implementeringen. Hovedsageligt interface og implementering af udleveret kode. Efter vi havde fået sat "skeletet" op, lavede vi unit test klasser, enkelte tests, Dernæst sat vi jenkins Web-Hook op (som vi havde glemt).
Vi har siden talt om at vi med fordel kunne have haft sat jenkin web-hook op til automatisk test og coverage fra start.

Fik Brugt Coverage til at finde enkelt code executions der ikke blev testet. Hertil forsøgt vi at lave test til de manglende, eksempelvis "ChargerControl.IsConnected".

Herudover påmindede Jenkins-coverage os om at konsol delene skulle testes igennem. Som det kan ses på vores "Coverage Report" på figur: 14, fra Jenkins. Kan det ses at der stadig mangler 4 funktioner i "ConsoleControl" at blive testers. Desværre fandt vi ikke en løsning på at teste dem.

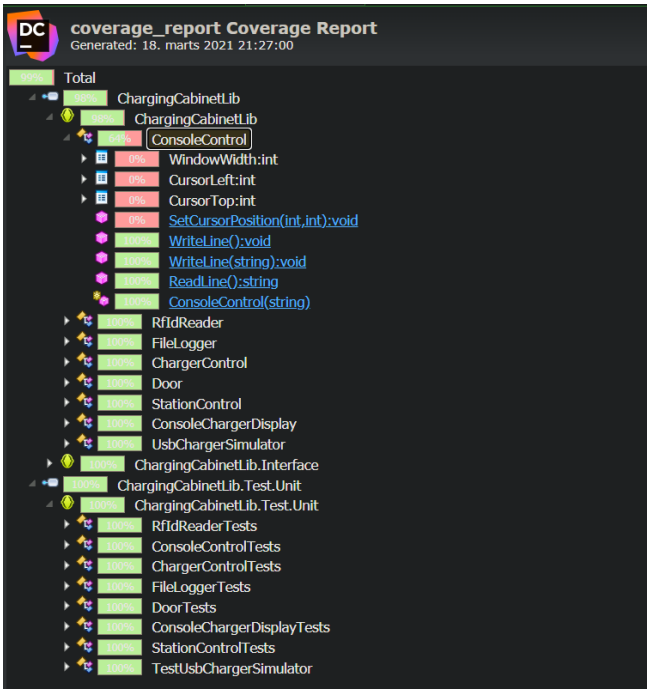

Figure 14: Coverage Report

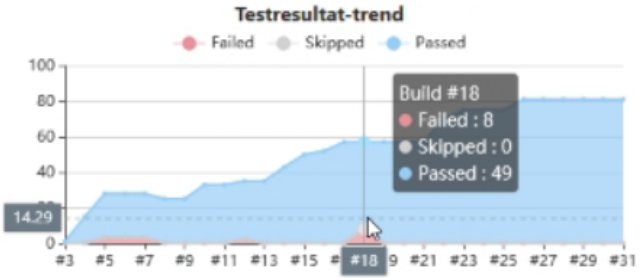

Figure 15: Jenkins TestResult

Vores Jenkins side kan finde på: 1[Jenkins-Link] ellers kan ses i indledningen.

3 Arbejdsfordeling

For at, fordele arbejdsbyrden bedst muligt, lavede vi en aftale om, at vi alle skulle kode og lave diagrammer på skift. Det var vigtigt, at vi alle tre fik mulighed for at prøve kræfter med alle aspekter af opgaven, så vi vidste, hvad der forgik i programmet. Vores plan lykkedes næsten efter hensigt, men det kunne ikke undgås, at byrden ikke blev lige fordelt, da vi alle tre har forskellige niveauer.

Vi har efterhånden fået en del erfaring i at bruge de redskaber, man bruger under opbyggelsen af et projekt som dette. Det var derfor relativt nemt og problemfrit at bruge Github til at dele filer og lægge dem op på Jenkins for at teste programmet. Der findes mange måder at bruge GitHub på, men det endte med at vi brugte Visual Studios UI til at uploade de nyeste versioner, når vi ændrede i filerne. Fordelen ved at bruge Visual Studio er, at man blot med enkelte tryk kan dele filerne med resten af gruppen, i stedet for at skulle ud af programmet hvergang.

Generelt observerende vi, at det gik glidende og problemfrit, men vi havde få episoder, hvor der opstod underlige fejl, som vi troede var merge-konflikt, men hvor der ikke var ændre noget kode, men ved at en file var blevet åbnet og lukke ned igen. Dette valgte vi at løse ved at slette Git fra den lokale pc, hvor efter det kunne hentes på ny.