Hand in number 2 – Integration Test

GENAFLEVERING!

Af Gruppe 8:

Max Barly Jørgensen	201401694
Nicholas Ladefoged	201500609

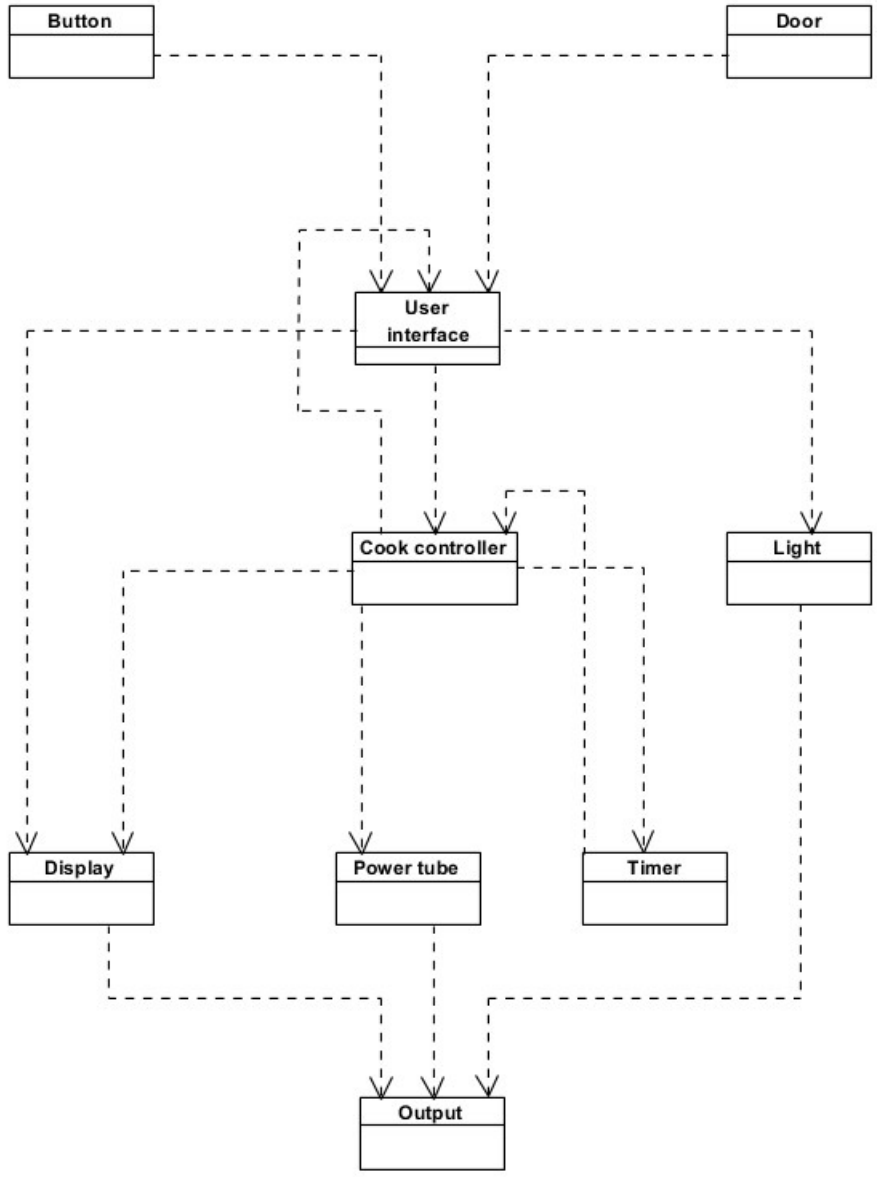
Jenkins Link: ci3.ase.au.dk:8080/job/TEAM8Handin2/

Github: https://github.com/nicho1991/SWTMicrowave

Indhold

Indledning:

Indledende vil vi se på dette dependency diagram som har til formål at give et overblik over hvordan microWave systemet er sat sammen, herefter vil vi beskrive hvordan vi har valgt at teste det. Vi har ud fra sekvensdiagrammet i opgaven analyseret os frem til dependency diagrammet i Figur 1.



Figur 1 Dependencys

Figur 1 Diagrammet har til formål at give et overblik over hvilke veje der skal testes når vi laver integrationstest. Dette diagram ligger altså grundlaget for hvilke strategier vi vælger når vi integrations tester.

Integrations test:

For at kunne sikre kommunikation imellem vores klasser, har vi integreret vores klasser således, at vi kan teste et givent output i forhold til et givent input. Hvilket giver os muligheder for at opfange eventuelle fejl der opstår i kommunikationen mellem klasserne.

Valg af strategi:

Efter analyse af sekvensdiagrammet samt vores eget dependency diagram, blev der enighed om at en top-down model ville give mest mening for os i denne test situation. Der skal bruges mange stubs til at udføre netop top-down, og dette er en af metodens negative sider, men med NSubstitute, er dette meget ligetil, og ikke et særligt stort problem. Med denne metode, kan vi også hurtigt se hvordan virkemåden i programmet fremkommer, frem for ved f.eks. bottom up, hvor vi tester laveste lag og op, giver et svært overblik for os.

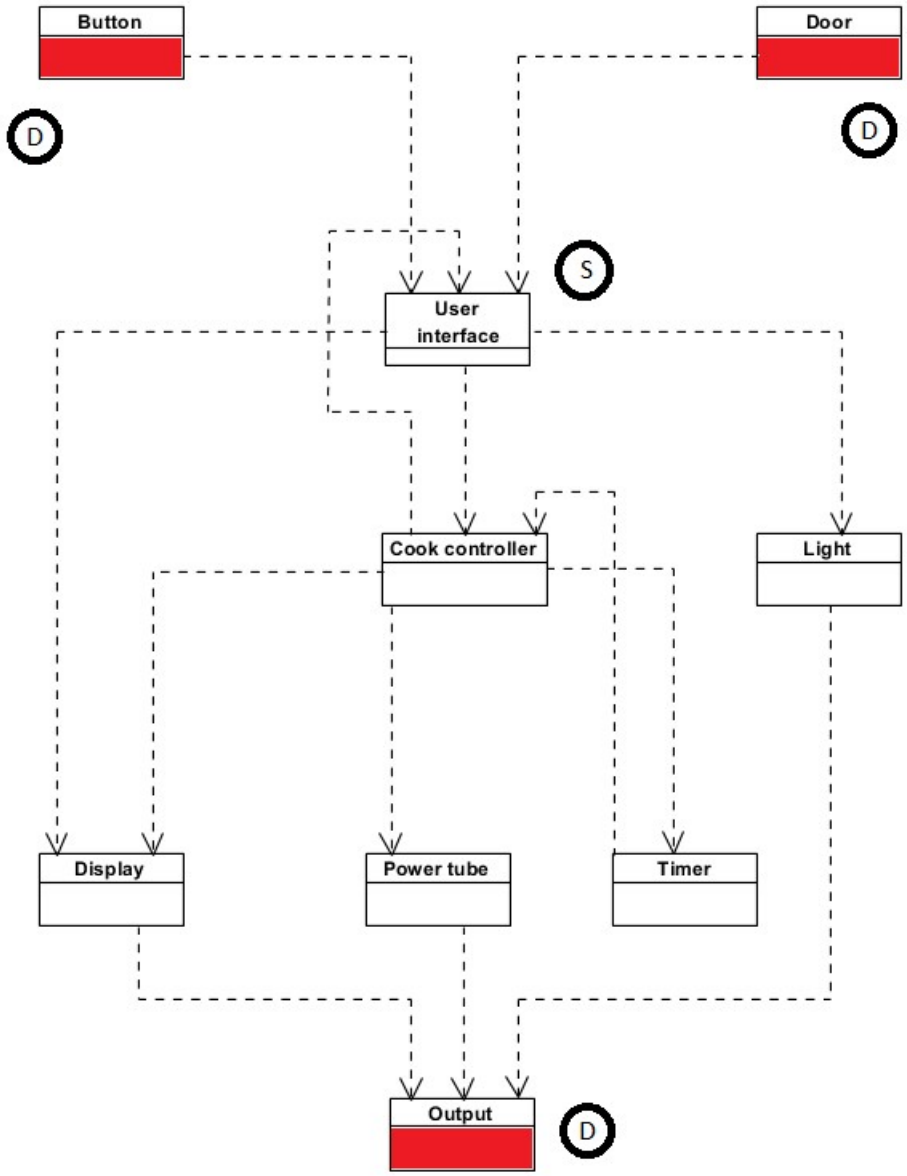
Herfra har vi lavet test scenarier der skal følge disse diagrammer. Dvs følgende af hvert diagram, er grundlæggende for et testfixture, som er overordnet tag for en test klasse. Hver klasse er hermed navngivet i rækkefølge af diagrammerne (integrationsTest1, IntegrationsTest2 og IntegrationsTest3)
Til sidst skal det nævnes vi har fulgt navngivning af Stubs, drivers og IUT ifølge diagrammer fra vores undervisning. Følgende:

1. Røde blokke markere klasser under test
2. Grønne blokke markere klasser der er testet
3. D markere driveren af testen
4. S markere stub i testen, her er brugt Nsubstitute til at stubbe.
5. Tykke streger markere hvilket niveau der testes på i forhold til røde blokke

Figur 2 – Unit test, viser hvilke dele der allerede er testet
Figur 2 er ikke en del af integrations tests, men giver et overblik over hvad der allerede er testet, i vores unit tests, og derfor ikke skal være en del af integrations testene. Dette kommer bedre til udtryk når vi starter fra Figur 3, som beskriver den første integrations test

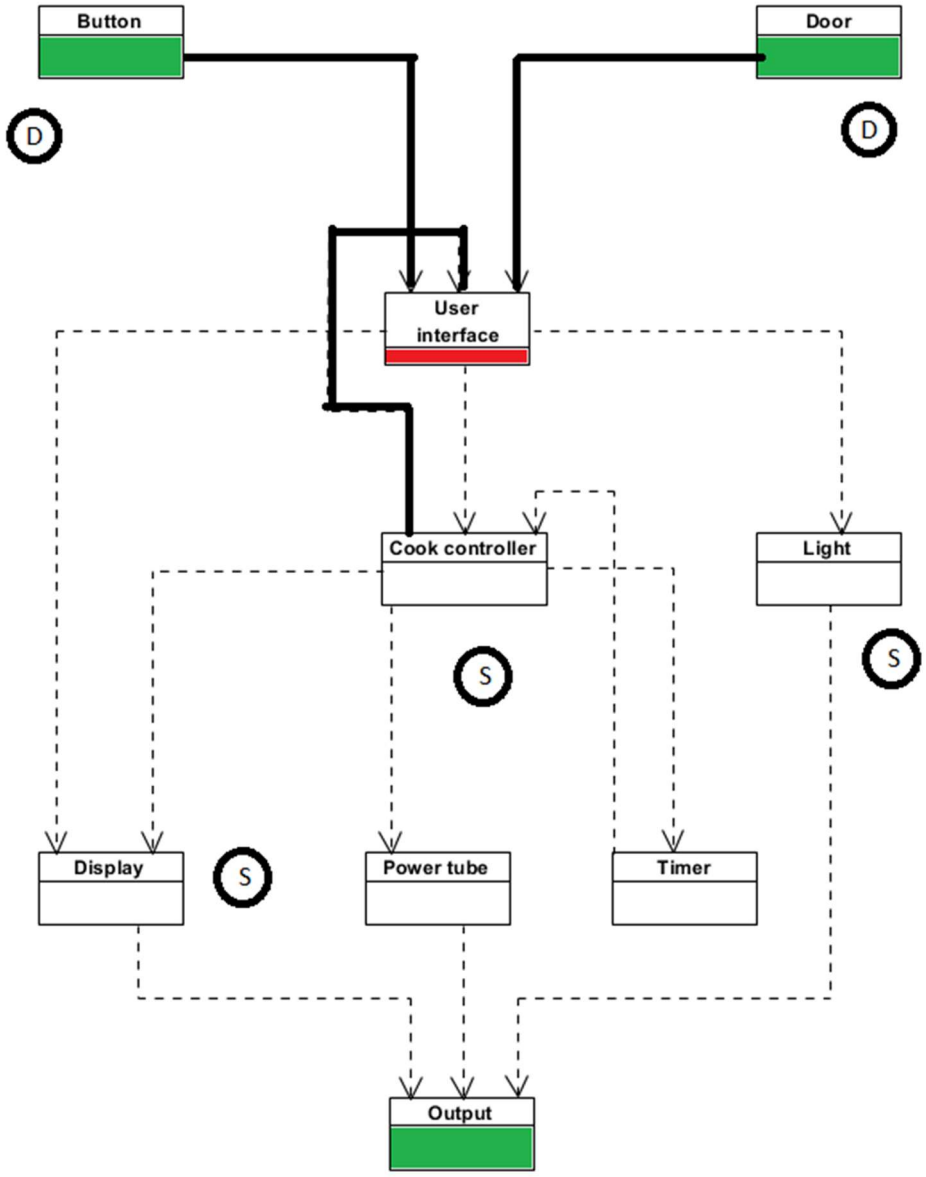
Figur 3-Integration test 1
Figur 3 viser vores første integrations test. Da vi følger top-down startes der med at teste fra user interface (under test) vi starter her da både button og door allerede er unit testet.
Vejen vi tester er fra button og door ned til User interface, igen gældende da vi følger top-down. Det interresante her er nok at cookcontroller, som egentlig ligger i et lavere lag end UI, også testes her. Dette skyldes at UI og CookController har dependencies begge veje, og dermed, skal i dette scenarie er user interface afhængig af cook controller.
Figur 4- Integration test 2
Figur 4 er vores næste integrations test. Her ses forskellen fra før at cook controller nu er afhængig af user interfacet, vi tester altså den anden vej, og et lag længere nede. Light er også under test her , da dens afhængighed ligger i samme lag som cook controller. Bemærk her at vi allerede nu begynder at teste mod Output, selvom den ligger i nederste lag, dette skyldes outputs direkte afhængihed af Light(under test).

Figur 5 - Integration Test 3
Figur 5 er vores sidste integrations test, eftersom output ikke har har afhængiheder af den længere nede. Denne test var omfattende da timers dobbelt afhængihed kan give vanskeligheder i forhold til at teste i et enkelt lag. Timers test, resulterede også i meget langsomme tests, da vi er nødt til at ”vente” på at timer er færdig i de konkrete scenarier vi tester.



Figur 2 – Unit test, viser hvilke dele der allerede er testet

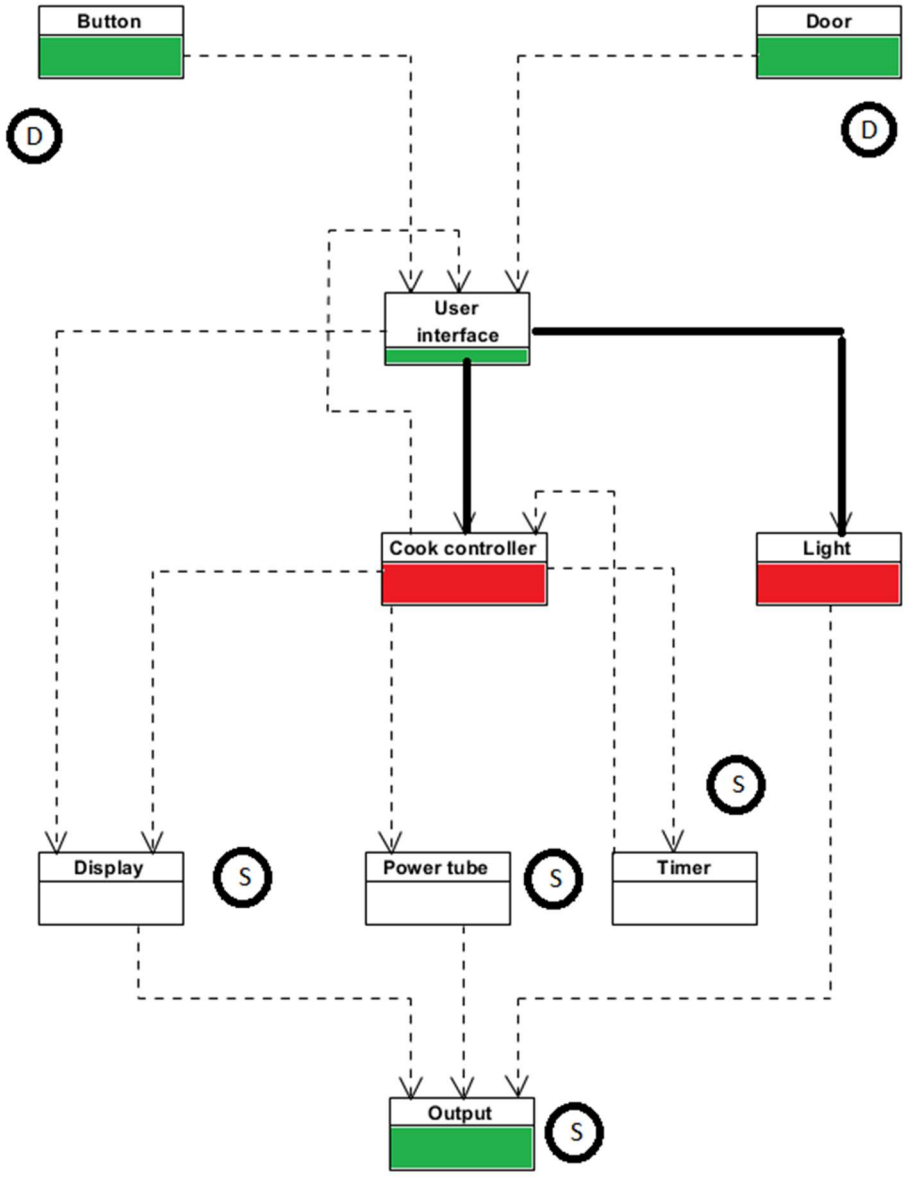
Figur 2 er ikke en del af integrations tests, men giver et overblik over hvad der allerede er testet, i vores unit tests, og derfor ikke skal være en del af integrations testene. Dette kommer bedre til udtryk når vi starter fra Figur 3, som beskriver den første integrations test



Figur 3-Integration test 1

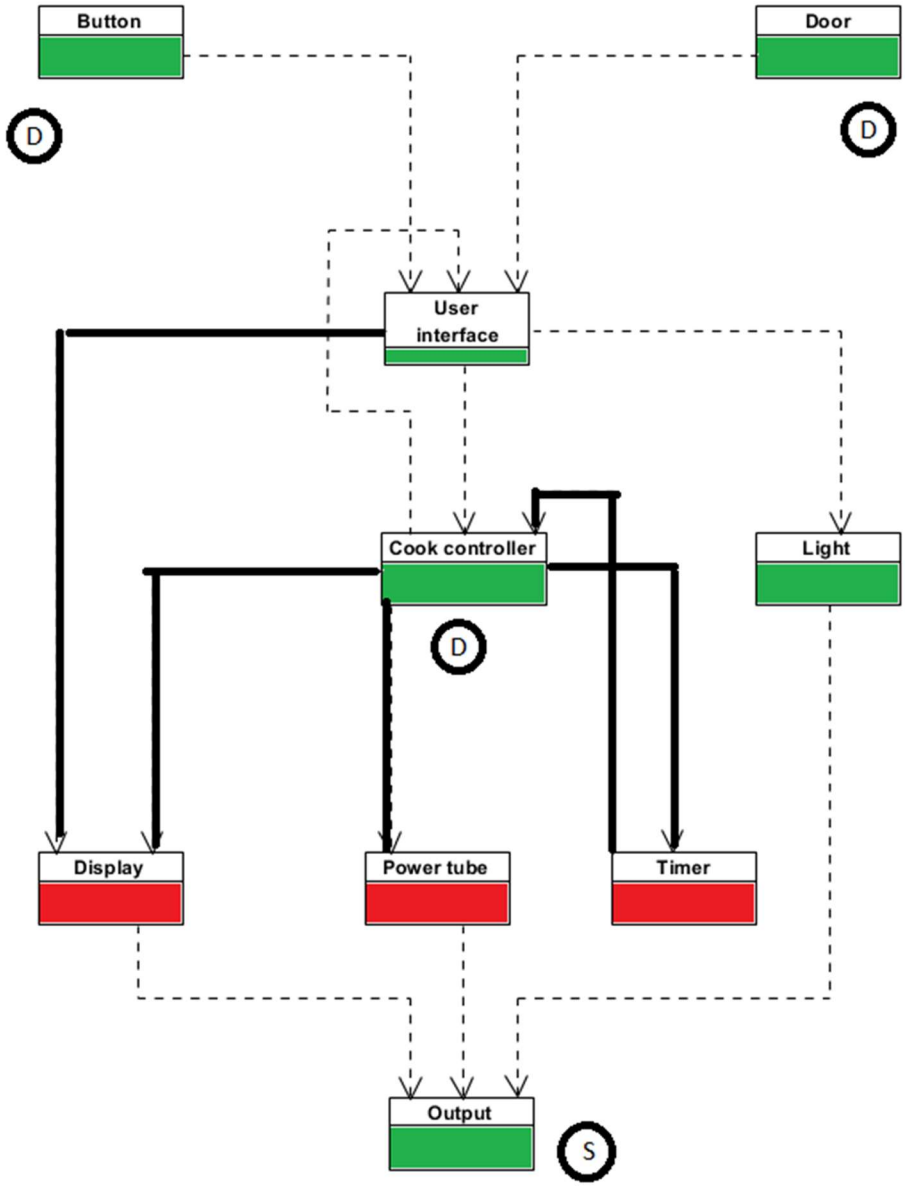
Figur 3 viser vores første integrations test. Da vi følger top-down startes der med at teste fra user interface (under test) vi starter her da både button og door allerede er unit testet.

Vejen vi tester er fra button og door ned til User interface, igen gældende da vi følger top-down. Det interresante her er nok at cookcontroller, som egentlig ligger i et lavere lag end UI, også testes her. Dette skyldes at UI og CookController har dependencies begge veje, og dermed, skal i dette scenarie er user interface afhængig af cook controller.



Figur 4- Integration test 2

Figur 4 er vores næste integrations test. Her ses forskellen fra før at cook controller nu er afhængig af user interfacet, vi tester altså den anden vej, og et lag længere nede. Light er også under test her , da dens afhængighed ligger i samme lag som cook controller. Bemærk her at vi allerede nu begynder at teste mod Output, selvom den ligger i nederste lag, dette skyldes outputs direkte afhængihed af Light(under test).



Figur 5 - Integration Test 3

Figur 5 er vores sidste integrations test, eftersom output ikke har har afhængiheder af den længere nede.

Denne test var omfattende da timers dobbelt afhængihed kan give vanskeligheder i forhold til at teste i et enkelt lag. Timers test, resulterede også i meget langsomme tests, da vi er nødt til at ”vente” på at timer er færdig i de konkrete scenarier vi tester.

Integrations Test plan:

Diagrammerne herunder

D: This module is included, and the one driven

X: This module is included

S: This modules is stubbed or mocked

Testing for Figur 3

Step #	Door	Button	UserInterface	CookControler	Timer	Light	Display	Power Tube	Output
1	D					X			
2		D	X				S		
3		D	X			S			
4		D	X	S		S	S		

Testing for Figur 4

Step #	Door	Button	UserInterface	CookControler	Timer	Light	Display	Power Tube	Output
1	D					X			S
2		D				X			S
3		D		X	S				
4		D		X	S			S	
5	D	D		X	S				
6	D	D		X				S	

Testing for Figur 5

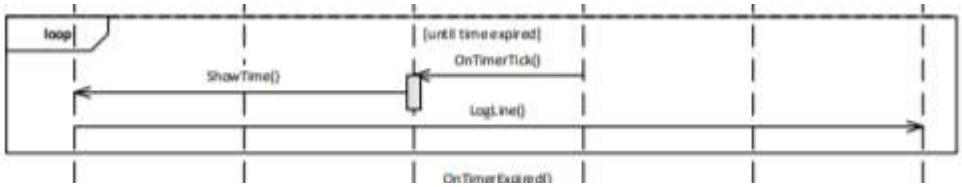
Step #	Door	Button	UserInterface	CookControler	Timer	Light	Display	Power Tube	Output
1				D	X				S
2	D	D			X		X		S
3	D	D			X				
4		D	X				X		S
5				D				X	S
6				D	X			X	S
7	D	D						X	S
8		D						X	S

Problemer:

Vores 3. unit test (se Figur 5) har givet os problemer i forbindelse med tråde der rendte ind over hinanden, derfor fejlede vores test tilfældigt. Vi har dog løst dette problem ved at ændre koden, se under Fejl Fundet.

Yderligere har vi haft problemer med 1 specifik test omkring timer i forhold til sekvensdiagrammet (se Figur 6) Testen i dette scenarie fejler. Tallene der skal vises på display hvert sekund, vises ikke korrekt. Vi tror det er et problem der ligger i koden, da outputtet simpelthen viser garbage. Fejlen viste sig at være fordi timeren tæller ned med 1000 sekunder i stedet for et, og dermed underlige tal vi ikke har forventet.

Sidste fejl er at consol ikke viser clear når timer løber ud. Denne fejl har vi ikke kunne løse, det resulterer i at TimerDisplay_timerDone_DisplayClear fejler.



Figur 6 problemet i timer test

Fejl fundet:

1: Fejlen er, at Timer indtastes i sekunder, men timeren omregner til millisekunder dette har vi løst således:

```
private void OnTimerEvent(object sender, System.Timers.ElapsedEventArgs args)
{
    // One tick has passed
    // Do what I should
    TimeRemaining -= 1;
    TimerTick?.Invoke(this, EventArgs.Empty);

    if (TimeRemaining <= 0)
    {
        Expire();
    }
}
```

Figur 7 time remaining bug 1000 til 1

```
[Test]
public void Start_TimerTick_CorrectNumber()
{
    ManualResetEvent pause = new ManualResetEvent(false);
    int notifications = 0;

    uut.Expired += (sender, args) => pause.Set();
    uut.TimerTick += (sender, args) => notifications++;

    uut.Start(2);

    // wait longer than expiration
    Assert.That(pause.WaitOne(2100));
    uut.Stop();

    Assert.That(notifications, Is.EqualTo(2));
}
```

Figur 8 Power tube

Figur 8 Der er en fejl her idet man i metoden turn on i powerTube klassen ikke omregner de indstillede watt til procent af samlede antal watt den kan indstilles på.

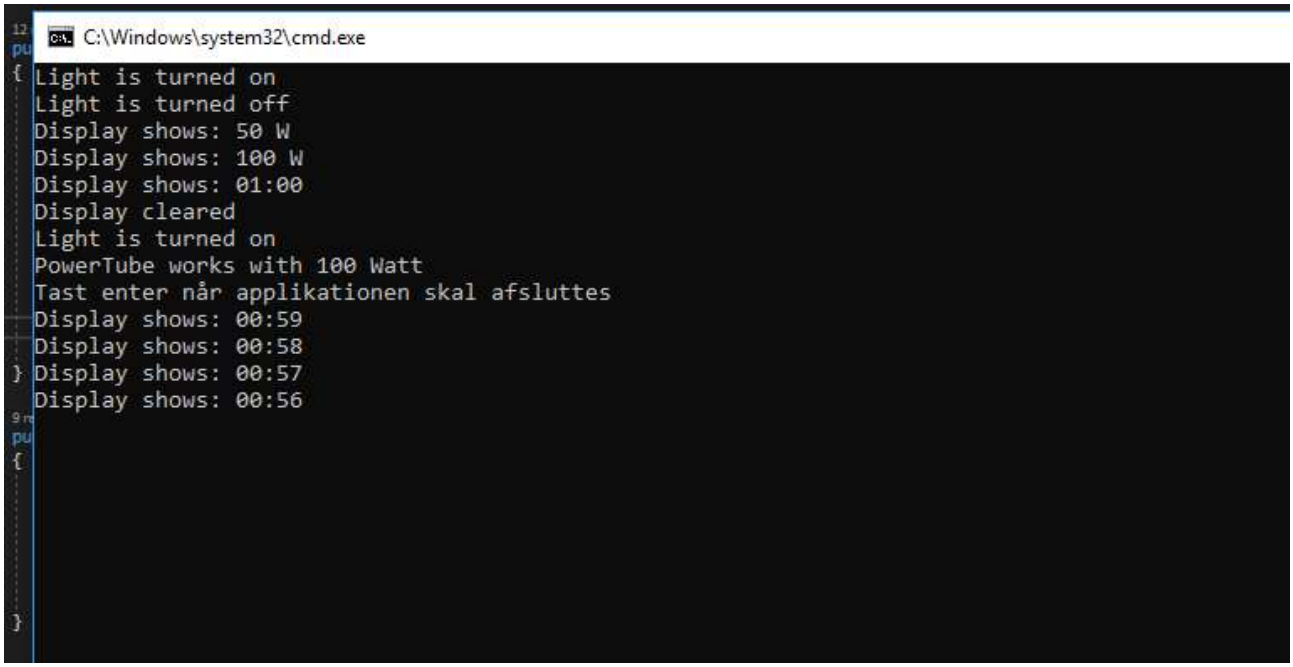
```
public void TurnOn(int power)
{
    if (power < 50 || 700 < power)
    {
        throw new ArgumentOutOfRangeException("power", power, "Must be between 50 and 700 (incl.)");
    }

    if (IsOn)
    {
        throw new ApplicationException("PowerTube.TurnOn: is already on");
    }

    myOutput.OutputLine($"PowerTube works with {power} Watt");
    IsOn = true;
}
```

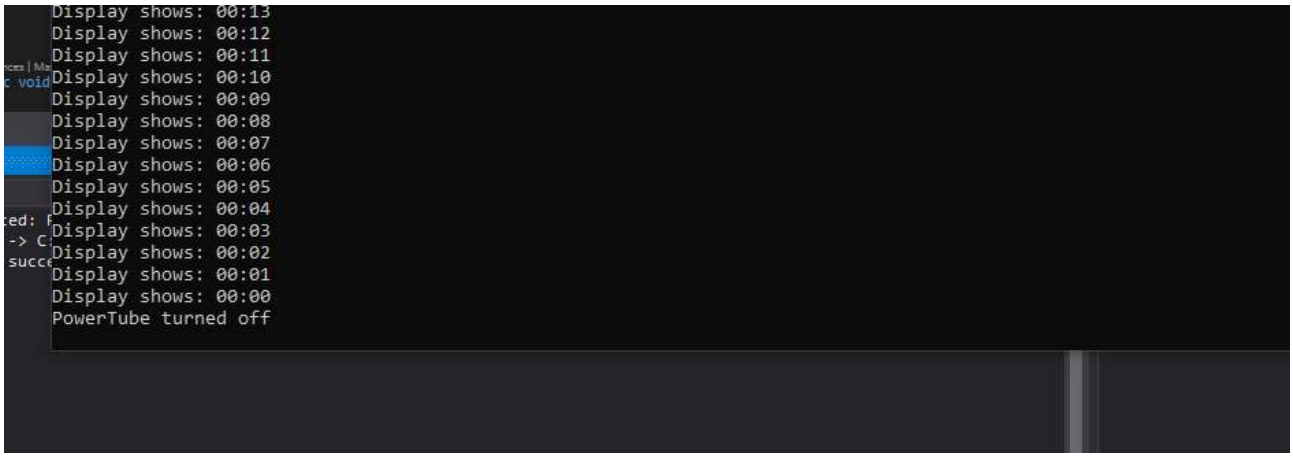
Figur 9 power fejl

Figur 9 fejlen her var at power som unit var sat op til at gå fra 1 til 100% men applikationen regner med 50 – 700 watt.

Sample applikation:

Vi har skrevet en sample applikation for at kører programmet som på normalvis. Her ses et consol udskrift når programmet køres:


C:\Windows\system32\cmd.exe
Light is turned on
Light is turned off
Display shows: 50 W
Display shows: 100 W
Display shows: 01:00
Display cleared
Light is turned on
PowerTube works with 100 Watt
Tast enter når applikationen skal afsluttes
Display shows: 00:59
Display shows: 00:58
Display shows: 00:57
Display shows: 00:56


Display shows: 00:13
Display shows: 00:12
Display shows: 00:11
Display shows: 00:10
Display shows: 00:09
Display shows: 00:08
Display shows: 00:07
Display shows: 00:06
Display shows: 00:05
Display shows: 00:04
Display shows: 00:03
Display shows: 00:02
Display shows: 00:01
Display shows: 00:00
PowerTube turned off