Figur 1- Source: https://www.toysrus.com/buy/toy-kitchen-appliances-tools/just-like-home-microwave-blue-ad11886-98393956

Software Test

MICROWAVE

HandIn2 | Gruppe 20 | 22-03-2018

Jakob Bonde Nielsen | 201607589
Kasper Juul Hermansen | 201607110
Stefanie Ruaya Nielson | 201605114
Rasmus Lund | 201310517

Indhold

Introduktion 2
URL JENKINS 2
GITHUB URL 2
Dependency tree diagram 3
Integrationsplan 4
Fundne fejl 4
Konklusion 5

Introduktion

Der skal i denne opgave udføres en integrationstest på et eksisterende givent system. I opgaven skal integrationstesten planlægges og herefter udføres. Integrationstesten planlægges ved hjælp af et dependency tree diagram, som viser afhængighederne mellem klasserne. Ud fra dette diagram laves en integrations plan, i samhørighed med dependency tree diagrammet.

Eventuelle fejl kommenteres og udredes. Testen opfølges af et program der afprøver systemet.

Det givent system består af software til en mikrobølgeovn.

URL JENKINS

Jenkins URL:

http://ci3.ase.au.dk:8080/job/TeamTyveMicrowaveH2/

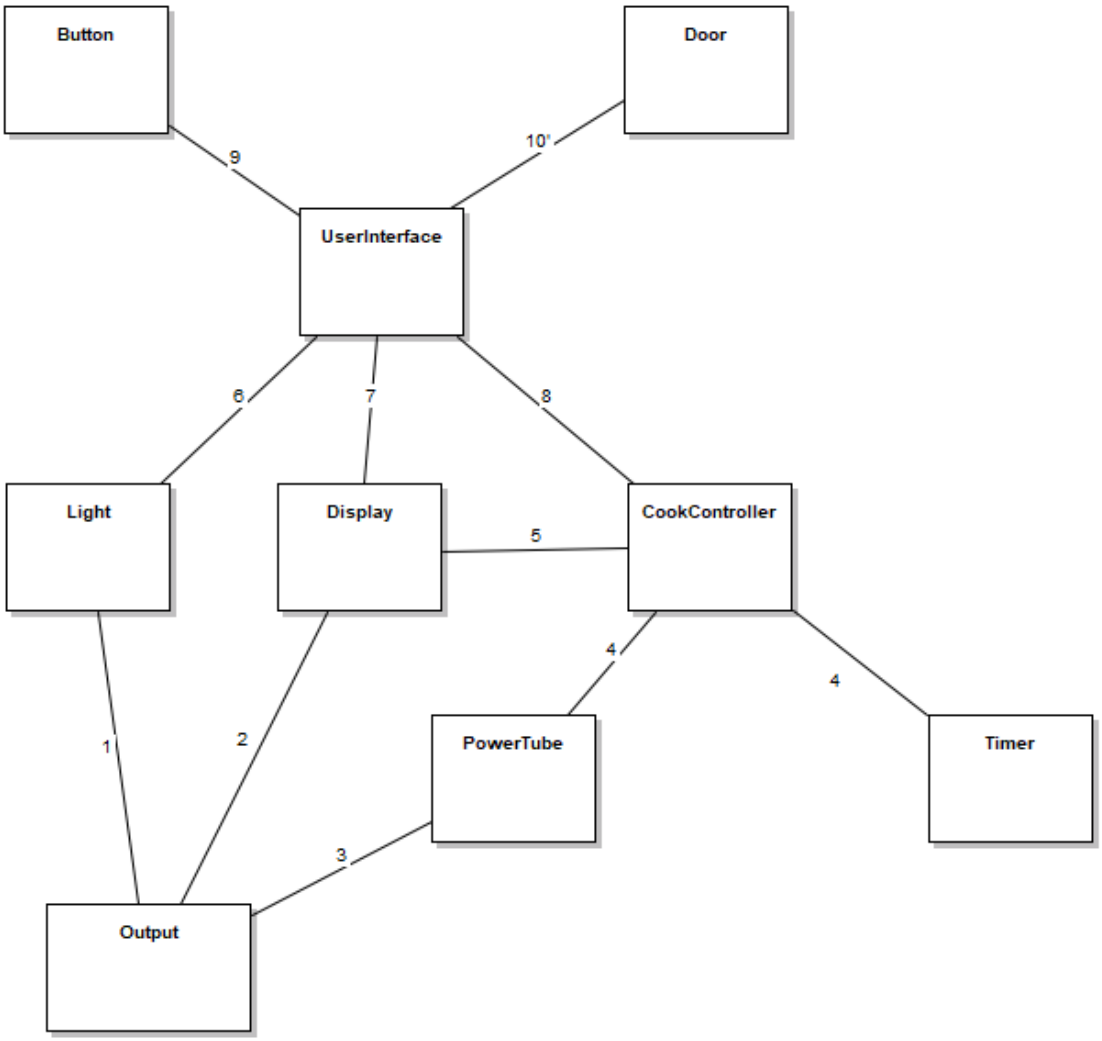
Jenkins URL for integration tests:

http://ci3.ase.au.dk:8080/job/TeamTyveMicrowaveH2-Integration/lastCompletedBuild/testReport/Microwave.Test.Integration/

GITHUB URL

Github URL: https://github.com/TeamTyve/MicrowaveHandIn2

Dependency tree diagram



Figur 2 - Dependency tree diagram for Microwave

På ovenstående figur ses et dependency diagram over Microwave systemet. Dependency diagrammet er udarbejdet ud fra den givne kode, og de givne sekvensdiagrammer. På de givne sekvensdiagrammet blev der observeret hvilke klasser der kalder hvilke metoder, og hvilke klasser der kender til hinanden. Træet er hierarkisk fra toppen og ned. Tallene ved siden af viser rækkefølgen af testene på interfacet mellem de 2 givne klasser. Dermed uddeler tallene integrationstesten i flere mindre bidder, og bevæger sig opad igennem træet. Det vil sige at vi har valgt at bruge bottom up pattern. I bottom up pattern startes der nedenfra, og herfra bevæges der opad mod toppen, hvor mere og afslutter med en fuld integrationstest. Dette har sine fordele og ulemper. En af de store ulemper er at man først vil opdage kritiske fejl i controller komponenter, relativt sent i integrationstesten. I det her tilfælde vil det betyde at fejl i controllere som CookController og Userinterface, vil blive opdaget senere i forløbet. En af

fordelene er at det ikke skal laves så mange stubbe og at interfacene er lette at dække.

Integrationsplan

Step	UI	CookController	Button	Light	Display	Door	PowerTube	Timer	Output
1				D					X
2					D				X
3							D		X
4	S	D			S		S	S	S
5		D			X		S	S	X
6	D	X	S	X	S	S	S	S	S
7	D	S	S	S	X	S			X
8	D	S	S	S	S	S			
9	X	S	D	X	X	X			S
10	X	S	S	X	S	D			S

Figur 3- Billede af integrationstest plan for Microwave

På ovenstående billede ses integrationsplanen for Microwave projektet. Integrationsplanen tager udgangspunkt i det tidligere viste dependency tree. Integrationsplanen viser hvilke klasser der er tilstede i de forskellige dele af integrationstesten.

D markere drivere, X er de klasser der indgår, S markere en stub.

Planen afsluttes ved step 10.

Fundne fejl

Følgende ændringer er foretaget i programmet:

```
public void TurnOn(int power)
        {
            if (power < 1 || 700 < power)  ////////////// CHANGED HERE ////////////
            {
                throw new ArgumentOutOfRangeException("power", power, "Must be
between 1 and 100 % (incl.)");
            }

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

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

        }
```

I ovenstående kodestykke er der blevet ændret fra procent til watt, som skulle være maks 700.

```
public void StartCooking(int power, int time)
        {
            myPowerTube.TurnOn(power);
            myTimer.Start(time * 1000); /////////////// CHANGED HERE /////////////
            isCooking = true;

        }
```

I ovenstående kodestykke er der ændret så timerens time bliver ganget med 1000. Dette er ændret da timerens opløsning ikke stemte overens med den ønskede.

```
public void OnTimerTick(object sender, EventArgs e)
        {
              int remaining = myTimer.TimeRemaining / 1000; // CHANGED HERE //
            myDisplay.ShowTime(remaining/60, remaining % 60);

        }
```

I ovenstående kodestykke er der blevet foretaget ændringer i det at TimeRemaining nu divideres med 1000.

Konklusion

Det kan konkluderes at integrationstest er et stærkt værktøj, og kan give stor værdi til et projekt. Integrationstest fungere godt sammen med continuos integration, og kan udføres løbende. Dette kan give en god fornemmelse af hvordan komponenterne spiller sammen med hinanden.
Integrationsplanlægningen i form af et dependency tree og en integrations plan, gjorde det mere overskueligt at lave integrationstesten. Dette hjalp meget da det ellers kan være svært at overskue en stor kodemængde.