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 | 11-04-2018

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

Indhold

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.

Jenkins URL

Jenkins URL:

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

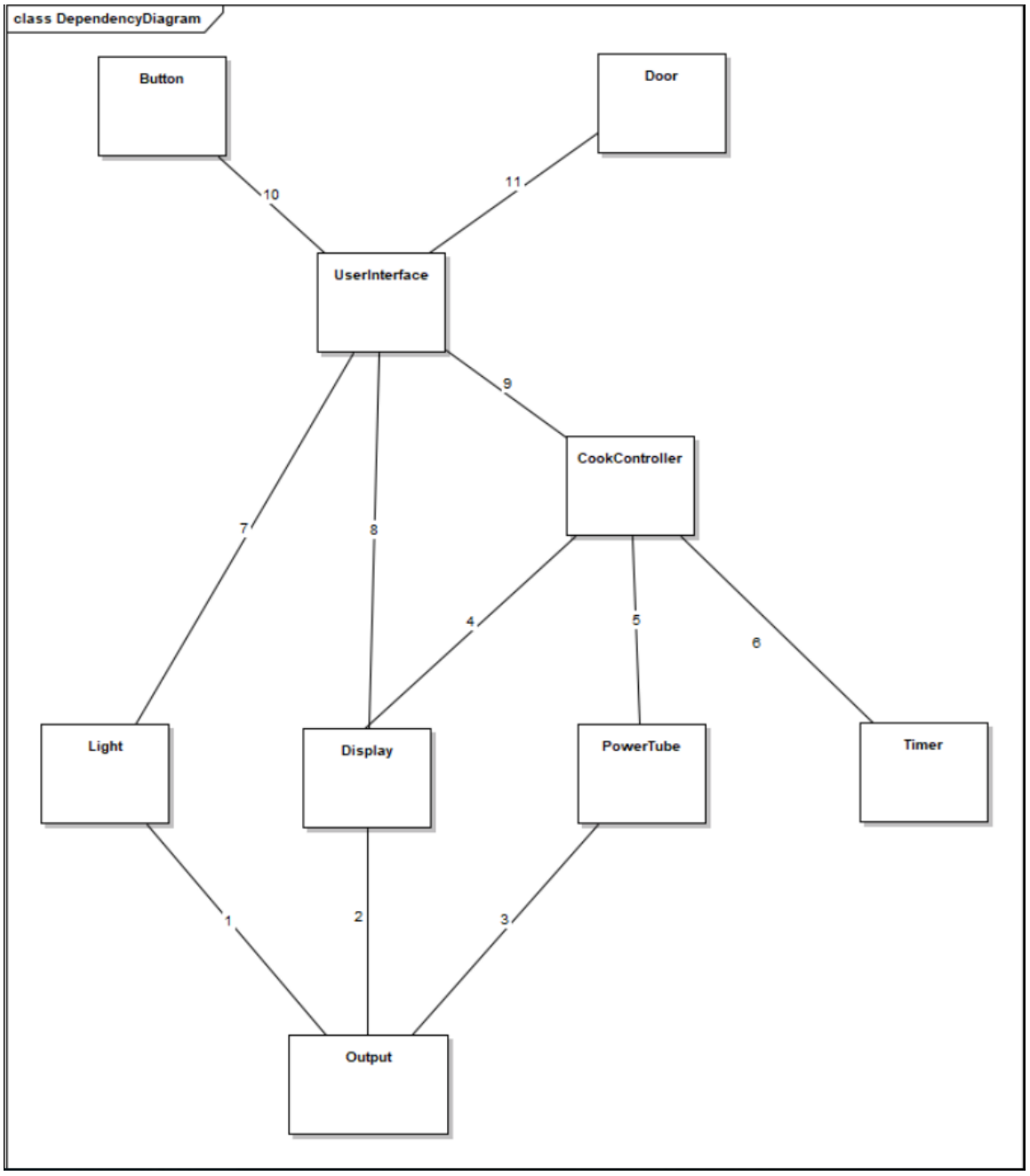
Jenkins URL for integration tests:

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 sekvensdiagrammer 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 imellem 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 nedefra, 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.

Generelt er der valgt at teste en mod en, og dermed er det ikke en sand bottom up. Konsekvensen af dette kan være at man ikke ser de fejl der bobler nedad. En sand Bottom-up er kendetegnet ved at hver SUT (System under test) dækker alle nedenstående afhængigheder. Vores løsning abstraherer sig her ved at vi har en rækkefølge på vores SUT afhængigheder, hvilken gør at vi ikke tester alle disse afhængigheder sammen, og derfor ikke kan opfange de events som forekommer ved f.eks. en fejl i test 5 ved test af 4. Som man normalt ville fange i en Bottom-up integration test.

For at kompensere for disse mangler, så er der tests som er afhængige af andre klasser for at kunne vise der output, et er Timer som skal benytte Powertube og Display. Dette kan sagtens løses med den "rækkefølge" vi bruger.

Vi har til test af interaktionen mellem Light, Display, PowerTube og Output brugt en stringwriter til at opfange de strenge som ville blive udsendt til konsollen. Dette er tildeles brugbart, men renfærdig gøre ikke at en systemtest som er påkrævet i dette tilfælde, da vi har med et eksternt element at gøre, såsom konsollen. Vi man nemlig ikke være sikre på at brugeren får vist det rigtige på konsollen, ved kun at teste det med en stringwriter.

Integrationsplan

Step	UI	CookController	Button	Light	Display	Door	PowerTube	Timer	Output
1				D					X
2					D				X
3							D		X
4		D			X		S	S	S
5	S	D			X		X	S	S
6	S	D			X		X	X	S
7	D	S	S	X	S	S			S
8	D	S	S	X	X	S			S
9	D	X	S	X	X	S	X	X	S
10	X	X	D	X	X	S	X	X	S
11	X	X	S	X	X	D	X	X	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 markerer den klasse der bliver drevet af driveren, X er de klasser der indgår, S markere en stub.

Planen afsluttes ved step 11.

Fundne fejl

Følgende ændringer er foretaget i programmet:

```
55.  public void TurnOn(int power)
56.      {
57.          if (power < 1 || 700 < power)                    /////////// CHANGED HERE ///////////
58.          {
59.              throw new ArgumentOutOfRangeException("power", power, "Must be between 1 and 700 W (incl.)");
60.          }
61.
62.          if (IsOn)
63.          {
64.              throw new ApplicationException("PowerTube.TurnOn: is already on");
65.          }
66.
67.          myOutput.OutputLine($"PowerTube works with {power} W");
68.          IsOn = true;
69.      }
```

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

```
41.  public void StartCooking(int power, int time)
42.      {
43.          myPowerTube.TurnOn(power);
44.          myTimer.Start(time * 1000); /////////// CHANGED HERE ///////////
45.          isCooking = true;
46.      }
```

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.

```
65.  public void OnTimerTick(object sender, EventArgs e)
66.      {
67.          int remaining = myTimer.TimeRemaining / 1000; // CHANGED HERE //
68.          myDisplay.ShowTime(remaining/60, remaining % 60);
69.      }
```

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

Test Applikation

Til denne opgave er der fremstillet to applikationer til test af selve programmet, dette er for at udføre en systemtest, som dækker hele systemet og udføres af brugeren, dette er f.eks. pga. af værktøjer til at kunne teste konsollen via. Automatiske tests. Der findes MicrowaveOven.Application projekt, som giver brugeren mulighed for at udleve usecase, og ”bruge” Microwave’s funktionalitet. Der ønskes også en automatisk test der gennemløber Microwaves funktionalitet, hvilket er beskrevet med MicrowaveOven.Automatic.Application denne test kører et enkelt scenarie igennem, hvor brugeren trykker på power, timer og start, efter nogle sekunder bliver der trykket på start igen og Microwave slukker, dernest åbner brugeren Microwave og Lukker Microwave, programmet terminere derefter.

Diskussion

Til denne handin skulle der udføres integration tests på et stykke software som vi havde fået udgivet, gruppen valgte derefter at vælge Bottom-up testing som det valgte værktøj til at gribe dette an, der er dog visse mangler i den udførte løsning. Gruppen har lavet en blanding af Sandwich og Bottom-up, da hele SUT ikke bliver testet sammen, men i stedet er delt op et mod en. Dette er derfor ikke en rigtig Bottom-up, og ikke kan siges at være en korrekt anvendelse af dette værktøj. Det giver dog muligheden for at undgå enorme driveren, som er svære at vedligeholde og opdatere, pga. Test bloat. Men konsekvensen er at de konkrete Klasser der bliver testet ikke altid bliver testet i kombinationen med parallelle forbindelser. Men i stedet bliver testet imod de konkrete klasser som spiller ind i SUT’s udfald. Så som CookController-Timer og de klasser som kræves for at kunne teste integrationen mellem disse.

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.