



MICROWAVE OVEN

I4SWT Gruppe 24



Navn	Studienummer	Email
Morten Bech	201604785	au568436@uni.au.dk
Alparslan Esen	201405877	au522394@uni.au.dk
Tri Nguyen	201610974	au564065@uni.au.dk
Berat Kaya	201610971	au572020@uni.au.dk

Github:

<https://github.com/i4dabber/Microwave>

Jenkins:

<http://ci3.ase.au.dk:8080/job/I4SWT24%20MicrowaveOven%20IntegrationTest/>

MAY 14, 2019

Table of Contents

Introduktion	2
Dependency Tree	2
Integrationsstrategi.....	3
Integrationsplan	5
Rettelser	5
Konklusion.....	6

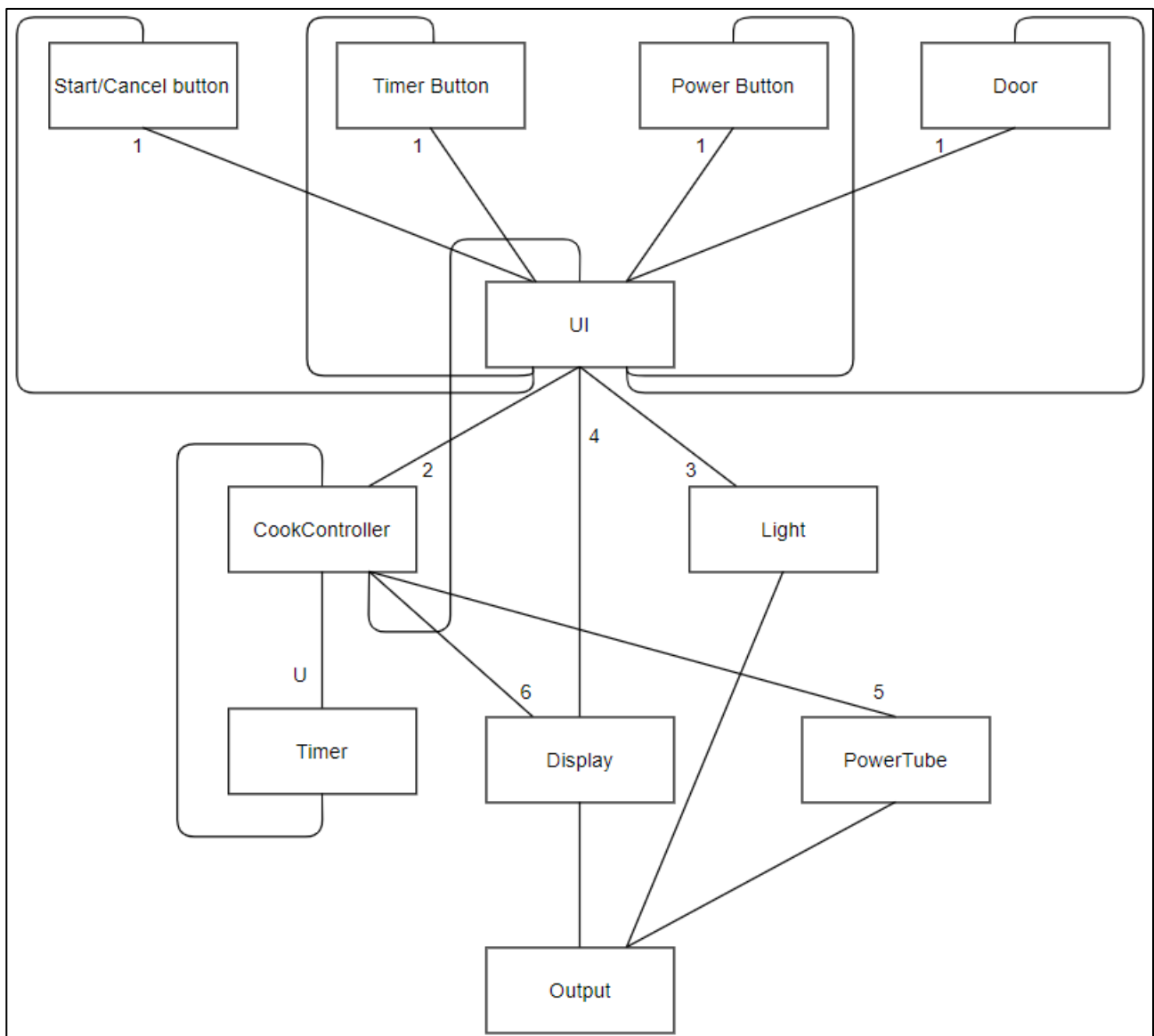
Introduktion

Til denne aflevering, har vi til opgave at lave integrations test for moduler/klasser. Vi har fået udleveret noget kode, der i forvejen er blevet unit-tested til at fungere korrekt. Disse funktioner benytter vi os af for at kunne udføre den endelige integrationstest.

Dependency Tree

Til denne øvelse har vi lavet dette dependency tree, der gerne skulle illustrere, hvordan vi har tænkt os at teste igennem

Først og fremmest har vi skitseret et dependency tree, der giver overblikket:

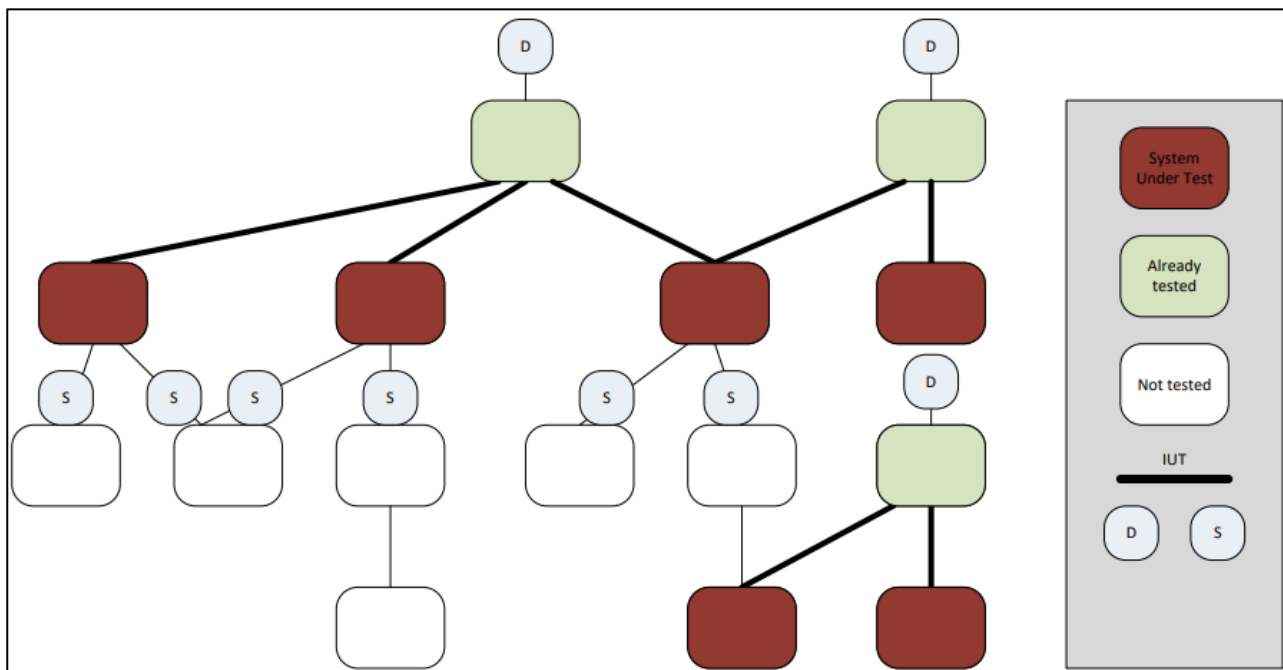


Figur 1 Dependency Tree for MicrowaveOven

Integrationsstrategi

Til at starte arbejdet med denne opgave begyndte vi alle at sætte os ind i den udleverede kode, inden længe forstod vi implementeringen og begyndte diskussionen omkring hvordan dependency træet skulle laves samt integrationstestene.

Det blev i gruppen besluttet at lave en Top-down integrationstest. Årsagen er at i Top-down integrationstesten faker man de nødvendige nedenstående klasser for at teste relationen mellem dependency-klasserne og stub-klasserne. Dette resulterer i at vi får hurtigere feedback på kontroller komponenterne (Buttons og Doors). Hvorimod i Bottom-down integrationen vil der kræves færre stubs.



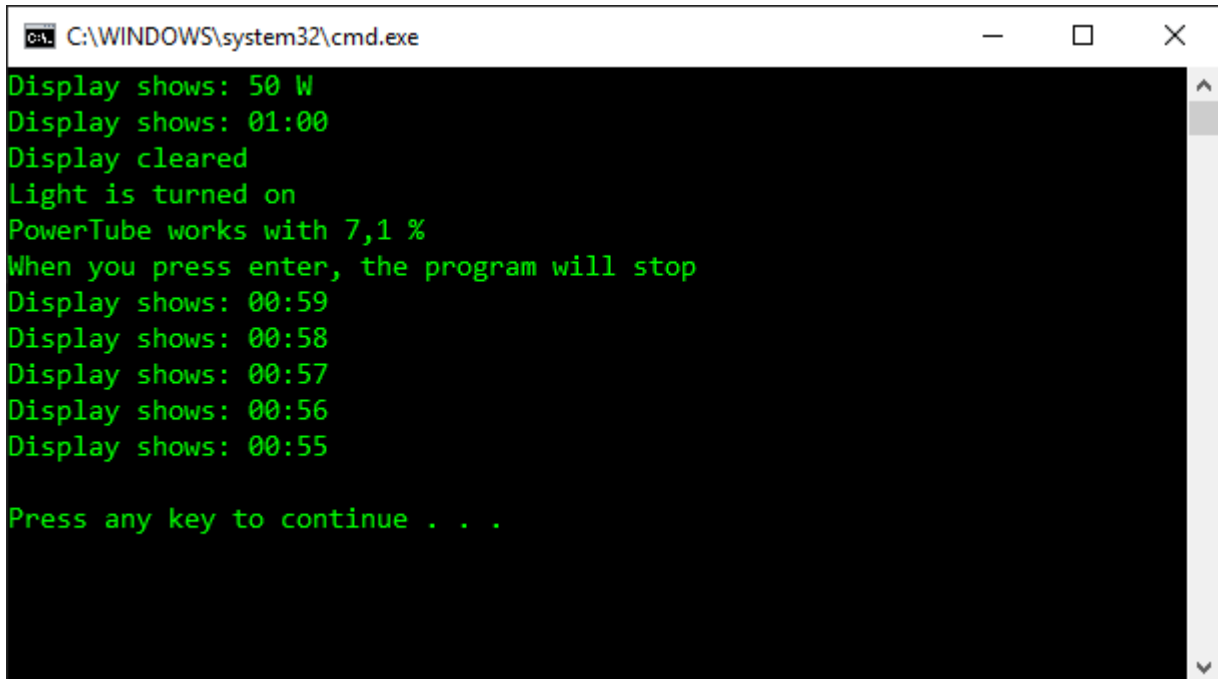
Figur 2 Illustration af en Bottom-Up Integrationtest-Pattern

Top-down integrationsmetoden går ud på at lave integrationstests fra toppen af det designede dependency tree, og derefter bevæge sig ned mod bunden. På den måde kan alle knapperne, samt døren, til alle integrationssteps inkluderes. Herved kan vi tjekke det forventede output på de klasser, der er under den testede klasse. Jo længere nede en klasse er placeret i dependency tree, jo mere afhængig er den af de andre klasser, så derfor er knapperne og døren lokaliseret øverst, da de kun kræver en bruger før de kan interageres med.

Udførelsen af et Top-down Integration foregår ved at starte med de øverste klassers integration med deres afhængige klasse på træet. I dette tilfælde vil det være knapperne og døren, og deres relation med User Interface. Derefter bevæger man sig så længere og længere ned i træet indtil man når et godt nok dække for hele koden.

Timer-klassen bliver ikke testet som en del af integrationstesten, da dens formål og handler er magen til CookController-klassen, så det er kun nødvendigt at teste for dem ene. Output-klassen er ikke taget med i integrationstesten, da dens eneste formål er at printe tekst ud på en terminal, og det giver ikke mening at teste om ting bliver skrevet rigtigt ud, når man udfører en WriteLine-kommando. Denne klasse bliver i stedet testes ved at oprette en applikation, og derved tjekke terminalvinduet om de rigtige ting bliver udskrevet på de rigtige tidspunkter. Applikationen sørger for at hver knap bliver trykket på én gang i rigtig

rækkefølge, hvorefter mikrobølgeovnen vil starte og begynde at tælle ned. Efter fem sekunder, bliver der trykket enter, hvilket vil få programmet til at stoppe. Denne applikationsterminal kan ses nedenunder:



```
C:\WINDOWS\system32\cmd.exe
Display shows: 50 W
Display shows: 01:00
Display cleared
Light is turned on
PowerTube works with 7,1 %
When you press enter, the program will stop
Display shows: 00:59
Display shows: 00:58
Display shows: 00:57
Display shows: 00:56
Display shows: 00:55

Press any key to continue . . .
```

Figur 3: Applikationens terminalvindue

Integrationsplan

Vi starter fra toppen, hvilket svarer til mikrobølgeovns knapper, dør og user interface. Integrationsplanen kan ses nedenunder i tabelform, og tager udgangspunkt i det designede dependency tree. Den benytter sig af 6 steps, som hver tester en forskellig klasse. I tabellen er alle klasser beskrevet, hvor de har fået tildelt et bogstav alt efter deres betydning til steppet i integrationstesten. Denne betydning kan være om klassen er unit under test, og den er inkluderet for at teste, eller om den er blevet faked. Grunden til at der bliver benyttet så mange fakes til de øverste integrationssteps, er fordi især UI-klassen har en stor constructor, der benytter sig af andre klasser som parametre.

Tabel 1 Integrationsplan for integrationstests

Step#	Output	Timer	Display	Power-Tube	Light	Cook-Controller	UI	Start/Cancel Button	Timer Button	Power Button	Door
1		S	S	S	S	S	T	X	X	X	X
2		S	S	S	S	T	X	X	X	X	X
3	S	S	S	S	T	S	X	X	X	X	X
4	S	S	T	S	S	S	X	X	X	X	X
5	S	S	S	T	S	X	X	X	X	X	X
6	S	X	T	X	S	X	X	X	X	X	X

Tabel 2 Integrationsplan ord-beskrivelse

T=	Modluet er inkluderet og drevet
X=	Modulet er inkluderet
S=	Modulet er faked

Rettelser

En af de ting vi har lagt vægt på er, at få rettet kode der ikke fungerer korrekt ad hensigten. De udleverede funktioner ville i sig selv godt kunne buildes, samt unit-testes korrekt, men de vil give de forkerte værdier til de tilsvarende integrationstests.

PowerTube's funktion TurnOn benytter sig af en parameter, int power, der afgør hvor meget effekt mikrobølgeovnen bruger når den er igangsat. Den er sat til at udskrive hvor stor en procentdel af den maksimale effekt den kører med på nuværende tidspunkt. Den originale kode fik funktionen til at udskrive parameteren, power, direkte som værende en procentdel.

Denne fejl blev fundet ved at prøve at køre kodens integere ud til dens begrænsninger, så effekten blev sat til 700 W. Dette gav en fejl, da TurnOn funktionen ville forstå den talværdi til at svare til 700% af mikrobølgeovns ydeevne, hvor der åbenlyst vil detekteres en fejl når man kommer over 100%.

Det vides at 100% power svarer til 700 W, så power-integeren kan omregnes til procent ved at gange med 100 (procent) og dividere med 700 (maks effekt), hvorefter der bruges den indbyggede funktion Math.Round til at afrunde tallet til den nærmeste tiendedel talværdi.

De tilsvarende Unit tests er blevet opdateret, så funktionen nu først er uden for rækkevidde når den modtager en power værdi større end 700 som parameter. Derudover blev det forventede output på terminalen ændret til ikke længere at være "50 %" men i stedet $(50 * 100 / 700)$, svarende til "7,1 %".

Timer's funktion OnTimerEvent bliver triggeret hver gang der er gået et sekund, og sørger for at displayet skal vise ét sekund mindre end den gjorde før. Den originale kode var ikke kontinuert om hvorvidt forløbet foregik i sekunder eller millisekunder, hvilket gjorde at for hvert sekund der gik i virkeligheden, ville mikrobølgeovnsens tid gå ned med 1000 sekunder. Displayet viste derfor "-15:-40", hvilket svarer til $60 - 1000 = -940$ sekunder.

timer.Start blev kaldt med en int time-parameter værende på 60, svarende til sekunder. Dens private integer TimeRemaining blev derefter også sat til 60, men den arbejdede i millisekunder i stedet. TimeRemaining er derfor blevet opdateret til også at arbejde i sekunder, så den også passer med de andre klasser.

Unittestene for Timer blev derefter rettet, så de korrekt benytter sig af 2 sekunder i stedet for 2000 sekunder når funktionen Timer.Start(int time) bliver testet for. Den indbyggede pause-funktion benytter sig stadig af millisekunder, så man må acceptere at tidsparameterene er en faktor tusind fra hinanden.

Konklusion

Vi som gruppe har arbejdet med at teste noget kode ved brug af integrationstest. Vi har brugt disse tests for at finde de steder hvor koden ikke fungerer som forventet, hvor der blev debugget for at finde præcist hvor i funktionerne fejlene var.

Ved at rette disse fejl, kunne vi opdatere Unit testene, så de ville fungere korrekt sammen med de opdaterede funktioner. Ved at bruge sekvensdiagrammen fra opgaven kunne der laves et dependency tree, som implementationen til integrationstestene kunne laves ud fra.

Der er i alt blevet lavet 6 integrationssteps, hvor forskellige klasser bliver enten testet, inkluderes eller faked. Der er i alt blevet implementeret 17 integrationstests, der er afspejlet af Top-down metoden, og som giver det korrekte resultat.