MICROWAVE OVEN INTEGRATIONSTEST

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/

APRIL 30, 2019

Table of Contents

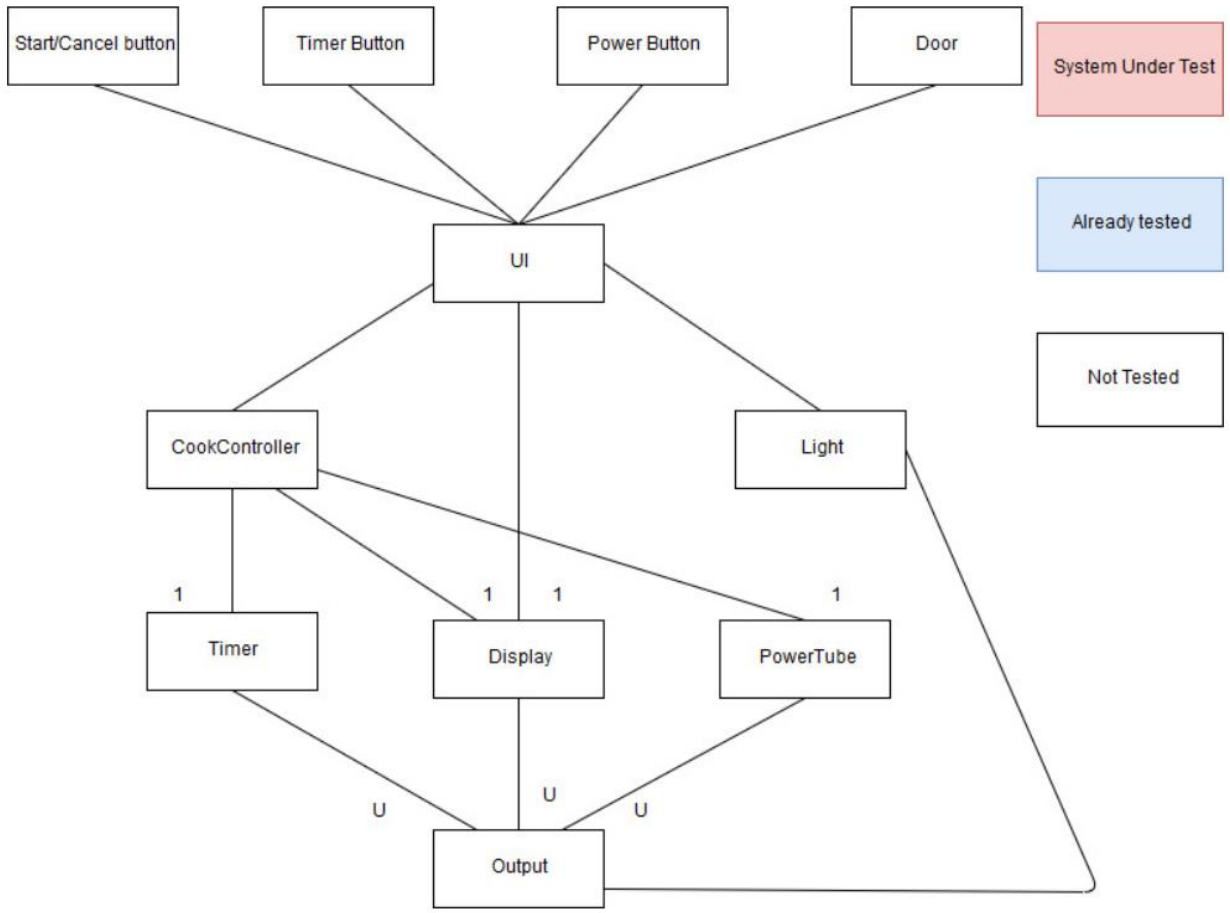
Introduktion

Til denne aflevering, har vi til opgave at lave integrations test for moduler/klasser. Vi har fået udleveret noget kode, som vi har ændret lidt ved for at få det til at passe. Dernæst har vi unit-testet for del elementer, 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:

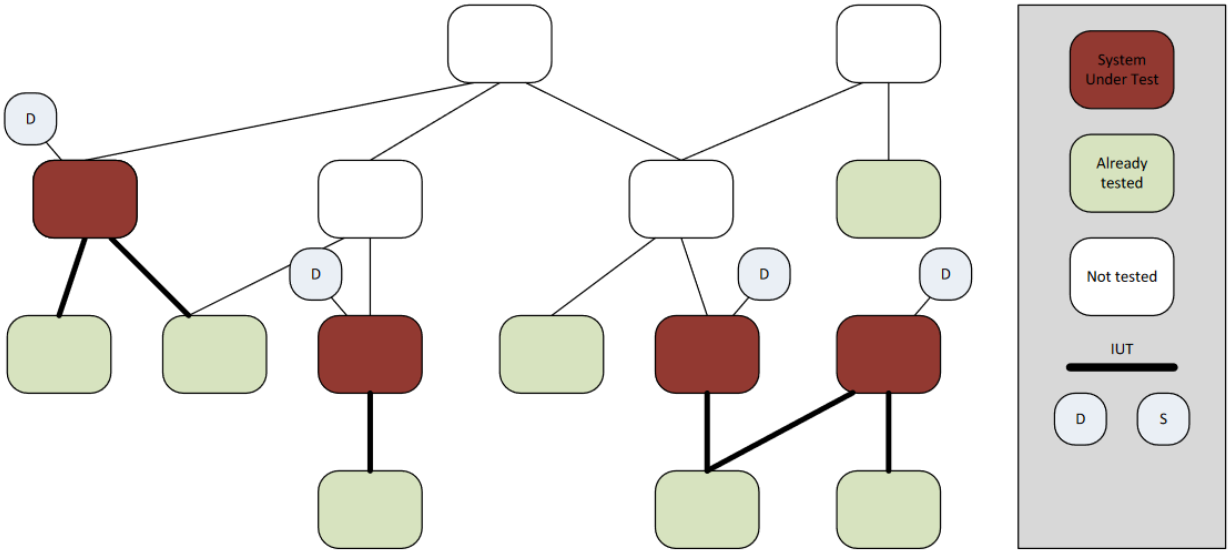


Figur 1 Dependency Tree for MicrowaveOven

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.

Vi i gruppen har besluttet os at lave et bottom-up integrations test, dette er fordi vi i gruppen syntes at det vil være bedre at lave mindre blackbox tests der bliver større indtil det er vi har hele systemet i en enkelt test.



Figur 2 Illustration af en Bottom-Up Integrationtest-Pattern

Bottom-Up Integrationspatternet går ud på at lave integrationstests fra bunden og lige så stille går op ad dependencytræet. Dette integrationspattern har vi valgt på baggrund af simpliciteten af implementationen af testene. Udførelsen af et Bottom-Up Integration forgår ved at starte med de yderste blade på træet, da vi allerede har fået den "endelige kode"[1] samt Unit-Testene til koden, så er de yderste blade allerede implementerede og testet. Det andet stadie af en Bottom-Up integration ville så være at gå et niveau op på træet, implementere, teste, og så integrationsteste det med det tidligere, siden den kode også allerede var implementeret og testet, skulle vi bare integrationsteste samarbejdet mellem bladene på træet og niveauet efter. Vi bliver så ved med at gå op ad træet indtil vi kommer til roden(eller rødderne) og får integrationstestet den.

[1] Den endelige kode bliver ændret senere i rapporten, da der var nogle bugs i koden.

Integrationsplan

Vi starter fra bunden, altså Output klassen unit testes, samt Timer, Display og PowerTube. Dernæst iterer vi op til vores CookController klasse, som vi laver flere intergrationstest på, hvor vi tjekker for om kommunikationen imellem klasserne fungerer.

Til selve integrationstestene har vi taget udgangspunkt fra de givne sekvens diagram, hvor vi f.eks. tester det sekventielle forløb imellem klasserne. F.eks. Hvis vi tager udgangspunkt i CookController integrationstesten, har vi fakede modulet for Output, som vi senere henne bruger. Vi kan i CookController finde metoden der hedder StartCooking(), hvor i denne metode, der kaldes der på de andre klasses metoder, hvor vi så til sidst checker om det er det forventede output vi skal have.

Herunder kan man se vores integrationsplan for de kommende tests. Dette er et udgangspunkt vi havde, som vi senere henne opdaterede for at få det til at passe.

Step#	Button	Door	User-Interface	Cook-Controller	Light	Display	Power-Tube	Timer	Output
1	S	S	T		S	X	S	S	S
2			S	T		X	X	X	S
3	X	X	X		T				S
4	X	X	X	X		T			S
5				X			T		S
6				X				T	S
7	X	X	X	X	X	X	X	X	T

Figur 3 Integrationsplan for integrationstests

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

Figur 4 Integrationsplan Legend

Rettelser

En af de ting vi har lagt vægt på er, at få rettet kode der ikke fungerer korrekt ad hensigten. Den udleverede kode ville i sig selv godt kunne build, men nogle af funktionerne og metoderne ville returnere forkerte værdier, som dog stadig ville kunne kører videre og til sidst vise en værdi som ikke var tænkt skulle være den værdi.

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 integers 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 mikrobølgeovnens 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 triggered 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 mikrobølgeovnens tid gå ned med 1000 sekunder. Displayet viste derfor "-15:-40", hvilket svarer til 60 - 1000 = -940 sekunder.

Måden fejlen blev fundet på var ved at debugge forløbet det sted hvor Timer-klassens funktioner blev kaldt, hvor det kunne ses at 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 sammen med breakpoints fundet nogle fejl i koden som vi har rettet.

Disse fejl som vi har rettet har så gjort at vi også er gået ind til Unit testene vi har fået udleveret af vores undervisere. Ved at så bruge sekvens diagrammet har vi lavet et dependency tree og implementeret vores integrations test.

Vi har lavet 14 integrationstest i alt hvor vi har brugt Bottom-Up metoden til at få hele vores funktionalitet testet. Disse specifikke test er beskrevet i vores Integrationstest kapitel.