Heisprosjekt rapport

av Lars Rønhaug Pettersen og Endre Vee Hagestuen

17. mars 2021

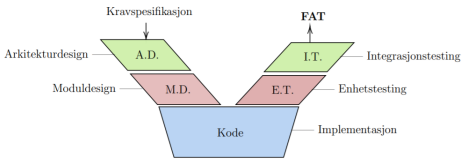
Labgruppe 3

TTK4235 Tilpassede datasystemer
Norges Teknisk-Naturvitenskapelige Universitet

Innhold

1 Kort om den pragmatiske V-modellen

I heisprosjektet er den pragmatiske V-modellen valgt som modell for systemutvikling. Denne er vist i figur 1.



Figur 1: Den pragmatiske V-modellen. Hentet fra utlevert prosjektoppgave, Kiet Tuan Hoang (2021).

Det aller første steget i den pragmatiske V-modellen er en analyse av produktkrav, gjerne fra en kunde. Resultatet av dette blir en kravspesifikasjon. I dette tilfellet er ikke denne analysen relevant, fordi kravspesifikasjonen er gitt i oppgaven. Se figur 6 for kravspesifikasjon.

Deretter kommer arkitekturdesign-delen, hvor man ut i fra kravspesifikasjonen deler inn systemet i moduler. Arkitekturen til systemet er i fokus, ikke hvordan hver modul skal implementeres. Kommunikasjon mellom modulene er også sentralt her.

I moduldesign-delen skjer implementasjonen av hver modul. Her kommer det frem hvordan innholdet i modulene realiserer ønsket funksjonalitet.

I bunnen av V-modellen skjer selve kodingen, hvor moduldesignet blir oversatt til kode. Dersom de forrige stegene er gjort grundig vil kodedelen bli lettere.

Deretter beveger man seg oppover på høyre side av V-en, hvor man første tester modulene hver for seg - enhetstesting. Her kan man avdekke mulige feil i enkeltmodulene før de settes sammen til et større system. Dette gjør eventuell feilsøking lettere.

Så kommer integrasjonstesting, hvor man tester om modulene fungerer som en helhet. Både under enhetstesting og integrasjonstesting er kravspesifikasjonen meget sentral, da den definerer kravene til systemet og dermed indikerer hva man skal teste.

Som en avsluttende test gjennomføres ofte en FAT-test (Factory acceptance test) i kontrollerte omgivelser, gjerne i produksjonslokalene.

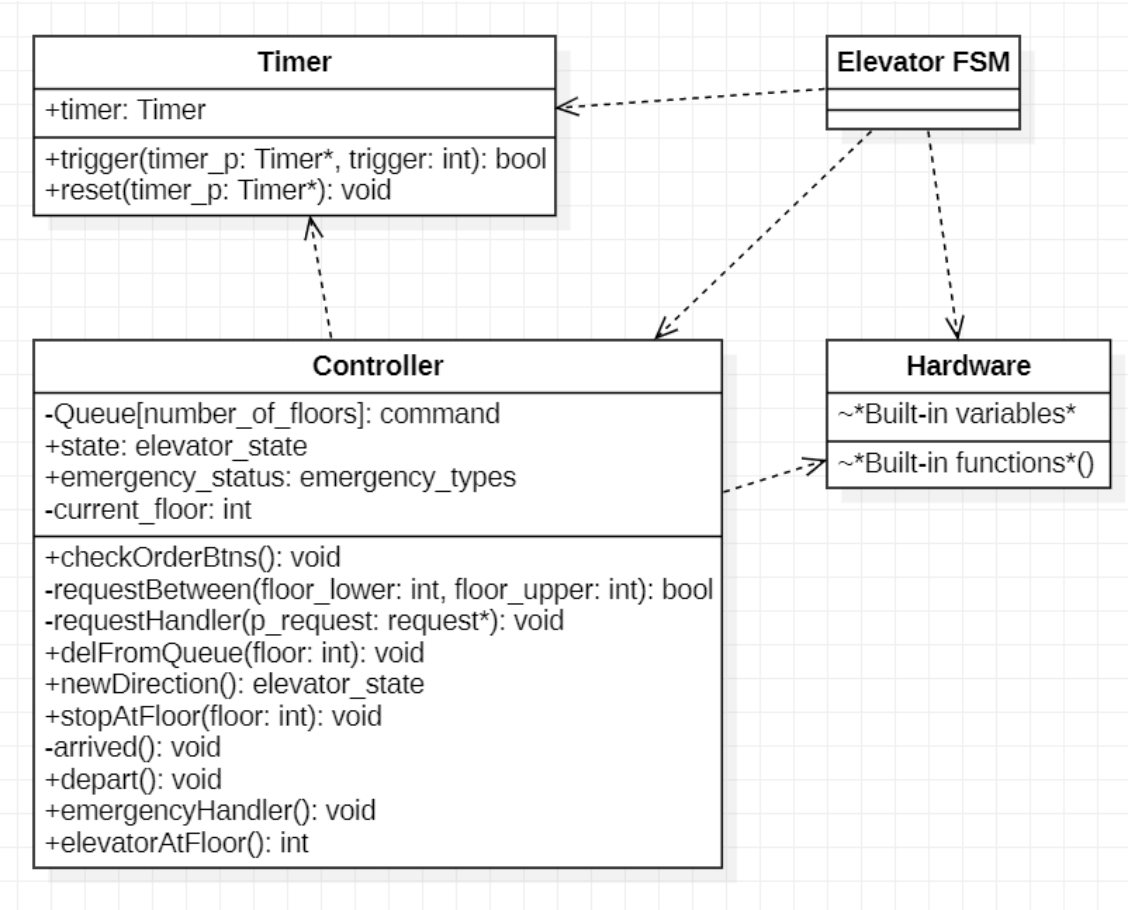
2 Overordnet arkitektur

2.1 Systemets arkitektur

Heissystemet består av følgende fire moduler:

- Elevator FSM (main.c)
- Hardware
- Controller
- Timer

Klassediagrammet i figur 2 illustrerer arkitekturen og viser relasjonene mellom modulene. Pilene i figurene markerer avhengigheter mellom modulene.



Figur 2: Klassediagram

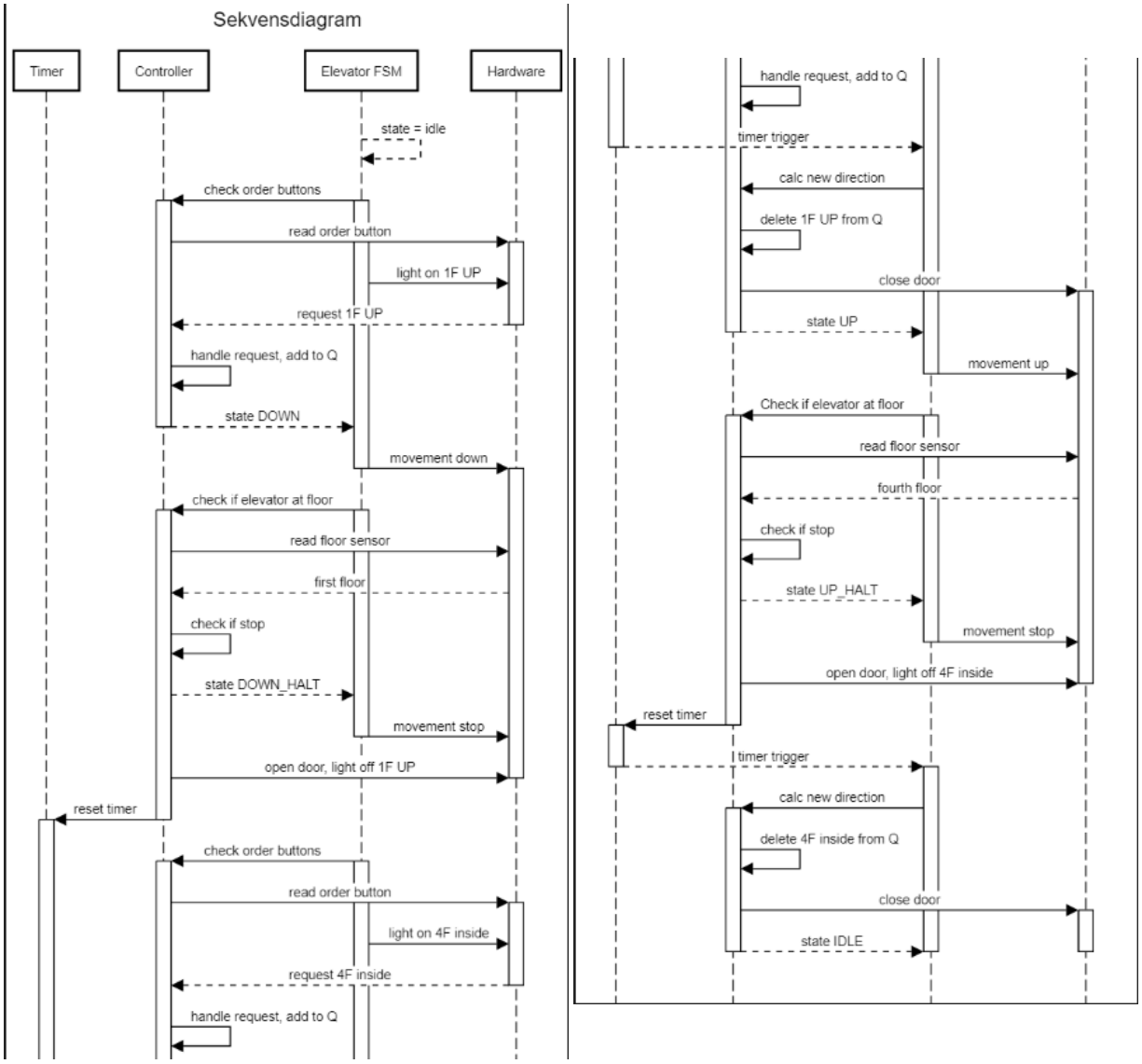
Med *built-in variables* og *built-in functions* menes variabler og funksjonalitet tilgjengelig fra hardware-filene som ble utlevert sammen med prosjektoppgaven.

2.2 Sekvensdiagram

Sekvensdiagrammet i figur 3 illustrer hvordan de forskjellige modulene fungerer sammen. Følgende sekvens er illustrert:

1. Heisen står stille i 2. etasje med døren lukket.

2. En person bestiller heisen fra 1. etasje.
3. Når heisen ankommer går personen inn i heisen og bestiller 4. etasje.
4. Heisen ankommer 4. etasje, og personen går av.
5. Etter 3 sekunder lukker dørene til heisen seg.



Figur 3: Sekvensdiagram (splittet vertikalt)

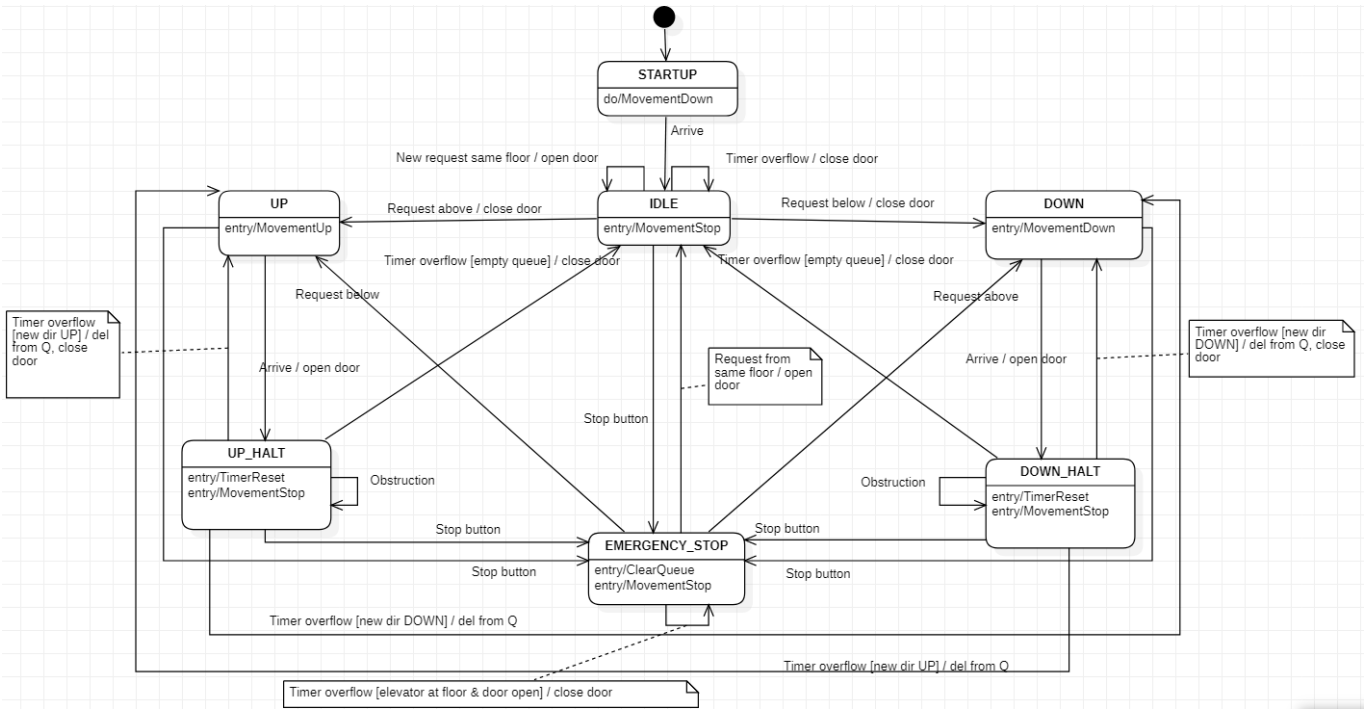
I sekvensdiagrammet ligger hovedfokuset på samhandlingen mellom modulene i den gitte situasjon, og hvordan heislogikken fungerer. Derfor er uvesentlige detaljer utelatt. Eksempel på dette er etasjeindikatorene. Dette er vesentlig for å dekke kravspesifikasjonen, men ikke for å forstå hvordan modulene samhandler for å oppnå ønsket heisoppførsel. Det er også verdt å merke seg at knappene og sensorene fra hardware polles kontinuerlig, men de er kun inkludert i diagrammet når det polles et aktivt signal.

2.3 Tilstandsdiagram

Heisens oppførsel defineres av totalt syv tilstander. Disse er som følger:

- STARTUP
- IDLE
- UP
- UP HALT
- DOWN
- DOWN HALT
- EMERGENCY

Hvordan heisen oppfører seg basert på disse tilstandene er vist i tilstandsdiagrammet i 4.



Figur 4: Tilstandsdiagram

2.4 Argumentasjon for arkitekturen

Arkitekturen er satt opp med mål om å få en kode som er delt inn i klare moduler. Dette gjør at man enkelt skal kunne bygge videre på, eller forandre programmet. Man ønsker altså at man kan bytte ut enkeltmoduler uten å måtte forandre på de restererende

modulene. For å få til dette er det viktig å fokusere på å gi modulene klare overordnede oppgaver.

Den overordnete oppgaven til timer-modulen er følgende:

- Start en timer.
- Det må være mulig å sjekke om en timer har utløpt.

Alle implementasjoner av en timer som oppfyller disse kravene vil kunne brukes i programmet.

Den overordnete oppgaven til hardware-modulen er følgende:

- Gi programmet tilgang til sensorinformasjon, styring av maskinvaren til heisen og lys.

Her også vil alle implementasjoner av en hardware-modul som oppfyller denne oppgaven kunne brukes i programmet.

Den overordnete oppgaven til controller-modulen er følgende:

- Holde kontroll på og bytte mellom states.
- Implementere køsystem og heislogikk.
- Behandle nødsituasjoner.

Her igjen vil alle implementasjoner av en controller-modul som oppfyller denne oppgaven kunne brukes i programmet.

Den overordnete oppgaven til Elevator FSM-modeulen er følgende:

- Bestemme heisoppførsel basert på states.
- Skrur på tilhørende lys når knapper trykkes inn.

Her vil også alle implementasjoner av en Elevator FSM-modul som oppfyller denne oppgaven kunne brukes i programmet.

3 Moduldesign

I arbeidet med moduldesign var skalerbarhet i fokus. Dette betyr blant annet at når man designer modulene skal det sørges for at systemet skal kunne brukes i bygg hvor man eksempelvis har hundre etasjer istedenfor fire.

3.1 Timer-modul

Denne modulen har en timer-struct som inneholder `clock_t` fra time.h. Programmet lager så en instans av denne. Dette gir mulighet til å opprette flere timere og øker med det skalerbarheten til programmet. Eksempelvis kan man bruke en ny timer til å kjøre heisen til første etasje hvis heisen har vært i idle i mer enn ett minutt. Dette kunne vært aktuelt i et kontorbygg på morgenkvisten, der de fleste skal oppover i bygget.

3.2 Controller-modul

Programmet holder kontroll på bestillinger ved hjelp av et array med dimensjon antall etasjer ganger antall ordretyper (*her: fire ganger tre*). Dette gir oversiktlig tilgang på bestillinger og gjør det enkelt å legge til bestillinger når bestillingsknapper trykkes og fjerne de igjen når de er ekspedert.

Controller implementerer heislogikk og inneholder funksjonalitet som avgjør hvilke etasjer heisen stopper på basert på bestillinger i køen og heisens tilstand. Det er også controller-modulen som bytter mellom de forskjellige tilstandene.

I denne modulen er det også implementert en rekke funksjoner for å oppnå ønsket heis-oppførsel. Med dette menes prosedyrer for ankomst og avgang ved en etasje. I tillegg har controller en funksjon som henter henter ut informasjon fra hardware for å orientere heisen.

En annen viktig oppgave til controller er å ta hånd om nødsituasjoner. Til dette har modulen en emergency status som hele tiden har kontroll på hvilken type nødsituasjon vi er i dersom vi er i en nødsituasjon. Tilstandsdiagrammet i figur 5 illustrerer dette. Dette er implementert for å få oppnå ønsket heisoppførsel spesielt ved nødstopp mellom etasjer.

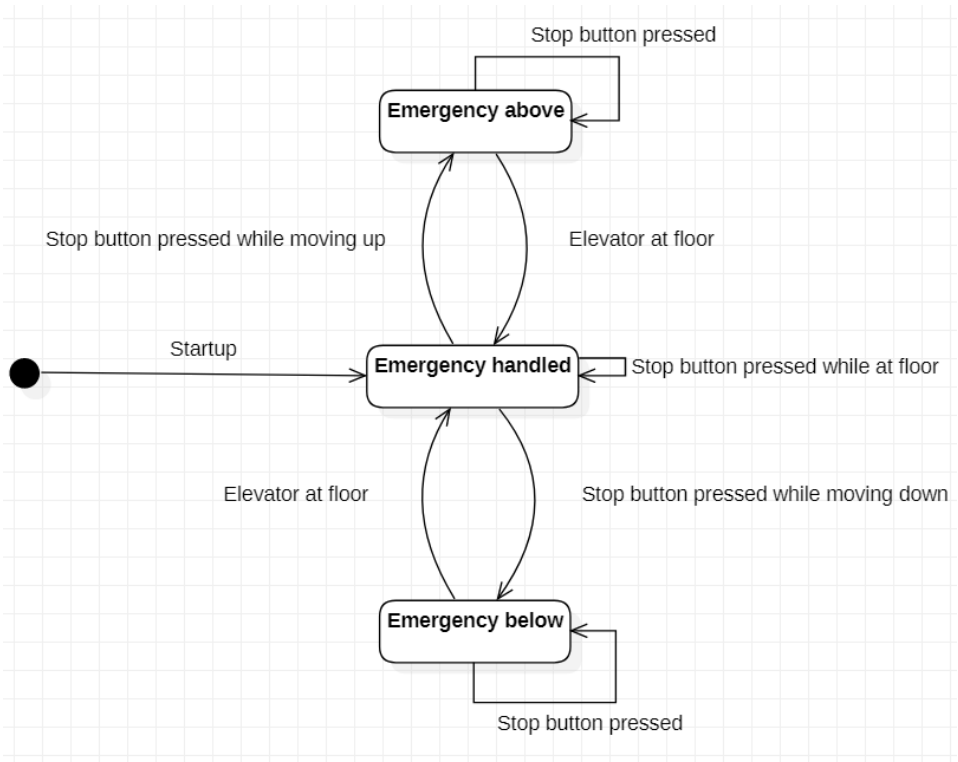
3.3 Elevator FSM-modul

Det er i Elevator FSM-modulen main switchen til programmet ligger. Det er her pådraget til heismotoren blir satt via hardware. Denne implementasjonen gjør at det kun er den gjeldende tilstanden som bestemmer om heisen går oppover, nedover eller står stille.

Det er også mainswitchen som avgjør hvilke funksjoner fra controller man kaller. Man kan dermed implementere forskjellig heisoppførsel med utgangspunkt i funksjonen tilgjengelig i controller-modulen.

3.4 Resulterende heisoppførsel

Følgende del er ment som et supplement til sekvensdiagrammet i figur 3 og tilstandsdiagrammet i figur 4. Resulterende heisoppførsel etter moduldesignet vil bli som følger:



Figur 5: Emergency tilstandsdiagram

- Heisen regner alltid ut en retning den skal bevege seg i, ikke en endelig destinasjon den skal bevege seg mot.
- Når heisen registrer en etasje vil den basert på køen og retningen bestemme om den skal stoppe eller ikke. Hvis en av følgende er oppfylt vil heisen stoppe:
  - Noen skal av i etasjen
  - Noen står utenfor heisrommet og skal i samme retning som heisen beveger seg.
  - Det er ingen flere bestillinger i de neste etasjene i bevegelsesretningen.
- Når ny retning regnes ut etter en bestillingsekspedering, vil følgende skje i prioritert rekkefølge:
  - Heisen finner bestillinger i tidligere bevegelsesretning og fortsetter i gitt retning.
  - Heisen finner bestillinger i motsatt retning og bytter retning.
  - Heisen finner ingen bestillinger og bytter tilstand til tilstanden idle.

4 Testing

4.1 Modultesting

4.1.1 Timer

Timer-modulen ble testet ved å opprette en instans av structen Timer, starte den, kalle trigger-funksjonen og printe til skjerm vha. printf() når timeren løp ut. Dette ble kontrollert ved bruk av egen klokke. Testen ble gjennomført flere ganger, og på denne måten ble det verifisert at timer-modulen bygget på clock() og `clock_t` fra biblioteket time.h fungerte som den skulle.

4.1.2 Hardware

Denne modulen var gitt som del av prosjektoppgaven, men ble likevel testet gjennom å manuelt sette opp to knapper for å styre retning på heisen i main.c. Valg av knapper varierte for å teste alle sammen. Samme prosedyre ble utført for å teste lys; knapper ble koblet opp direkte til lys i main.c. På denne måten kunne knapper, motordrift og lys verifiseres visuelt.

4.1.3 Controller

Enhetstesting av Controller bestod av å teste funksjoner individuelt for å ha enklest mulig feilsøking dersom feil skulle oppstå. Dette inkluderer å teste å legge til/slette bestillinger fra køen, teste retningsutregning ved ulike scenarier og å teste at heisen stopper ved riktige etasjer. I tillegg må støttefunksjoner som detektering av bestillinger i et etasjeintervall testes.

For å vise hvordan testingen ble utført brukes retningsutregning som eksempel. Dette ble testet slik:

1. Initialiser heisens tilstand i main.c.
2. Initialiser køen, og legg til et sett bestillinger
3. Kall på newDirection og se om funksjonen returnerer forventet tilstand

Denne prosedyren ble gjentatt med ulike kombinasjoner av inputs for å verifisere at logikken var robust.

4.2 Integrasjonstesting

Integrasjonstesting ble gjennomført med utgangspunkt i kravspesifikasjonen. Krav for krav ble gjenomgått med tilhørende testscenarioer, og ved observasjon av resultat ble eventuelle feil rettet opp. Fordelen med denne teststrukturen er at siden de individuelle modulene allerede er testet, vet man at eventuelle feil ligger i koblingen mellom dem. Gjennom denne prosedyren ble systemet verifisert.

4.3 Testscenario H2

Følgende del er ment som et eksempel på et testscenario som dekker en spesifikk kravspesifikasjon. H2 er brukt som eksempel, se figur 6. Scenariet er som følger:

1. Heisen står i ro i 1. etasje. Det har den gjort i lang tid.
2. Heisen får en NED-bestilling fra 3. etasje.
3. Før heisen har kommet til 3. etasje, får den en NED-bestilling fra 4. etasje.

Etter krav H2 skal heisen gå forbi bestillingen i 3. etasje når den er på vei opp, fordi bestillingen er i motsatt retning av heisbevegelsen. Ventet oppførsel er dermed som følger:

1. Heisen går til 4. etasje, venter i tre sekunder.
2. Heisen går til 3. etasje.

Videre oppførsel er avhengig av eventuelle bestillinger som blir gjort.

Observert oppførsel svarte til ventet oppførsel, og testen ble dermed godkjent.

5 Diskusjon

En mulig svakhet i designet er at den delen av koden som er ansvarlig for å kalkulere ny retning etter en bestilling er ekspedert returnerer en *retning*, ikke en *destinasjon* - dette gjør at man også krever logikk for å bestemme om heisen skal stoppe ved en etasje den passerer, istedenfor å kjøre helt til den når en forhåndsbestemt destinasjon. Dersom man på et senere punkt ønsker en helt ny funksjonalitet for prioritering av bestillinger, må den aktuelle utvikleren altså endre på både retningslogikken og stopplogikken for å oppnå ny ønsket heisoppførsel.

På den andre siden ville utvikleren hatt tilgang til mange gode støttefunksjoner i et ellers oversiktlig arkitekturdesign. Det er dermed fullt mulig for en utvikler uten kjennskap til

hele systemet å implementere ønsket heisoppførsel. Dette er en styrke med løsningen, fordi det gjør det lettere å vedlikeholde koden.

Et forslag ville vært å la funksjonen som returnerer ny retning heller returnere en destinasjon. Da blir stopp-logikken mye enklere; man behøver bare sjekke om etasjen man passerer er lik etasjen man har som destinasjon. En ulempe med destinasjons-løsningen er derimot at man fortløpende må oppdatere destinasjonen dersom man får en ny bestilling. Dette er det ikke behov for i løsningen valgt i dette prosjektet, hvor nye bestillinger simpelt legges til i køen dersom heisen er i bevegelse, og stopp-logikken tas etasjevis når heisen passerer nye etasjer.

Et annen forslag til endring er å slå sammen logikken i requestHandler-funksjonen og newDirection-funksjonen. Med valgt design i dette prosjektet er situasjonen at handleren kalles når en ny request polles, og legger den til i køen. *I tillegg* har den muligheten til å sette i gang heisen dersom heisen er i tilstandene idle eller emergency stop (tilfeller hvor køen er tom, og ny bestilling er eneste gjeldende bestilling). Retningsfunksjonen kalles etter heisen har stoppet i 3 sekunder i en etasje, og er klar for å ekspedere en ny bestilling. Oppsummert switcher disse to funksjonene på samme variabel (heisens tilstand), *men har ingen overlapp* i switch-casene. En konkret foreslått endring er følgende:

1. Flytt switch-logikken fra request handleren til new direction.
2. Kall new direction hver gang handleren har lagt til en ny bestilling i køen.

Dersom man velger å slå sammen disse funksjonene, vil request handleren ikke ha noen oppgaver utover å legge til bestillinger i køen. Som indikert i listen oppgitt over ville en mulighet vært å kalle new direction hver gang handleren har lagt til en ny bestilling i køen. På den positive siden ville dette samlet logikken på ett sted (new direction), men på den andre siden ville det blitt en svært tung funksjon for å behandle alle mulige scenarier. To scenarier for å videre drøfte endringsforslaget:

1. Heisen er i bevegelse fra 2. etasje til 3. etasje med ingen andre bestillinger i køen. Heisen får en bestilling til 1. etasje.
2. Heisen har ankommet 3. etasje, og har stått der i 2 sekunder med ingen bestillinger i køen. Heisen får en bestilling fra 1. etasje.

Ønsket oppførsel:

1. Heisen legger til ny bestilling i køen, fullfører ekspedering av 3. etasje, og kjører deretter til 1. etasje.
2. Heisen legger til ny bestilling i køen, venter ett sekund til i 3. etasje, og kjører deretter til 1. etasje.

Endringsforslaget vil føre til korrekt heisoppførsel i det første scenariet. Grunnen til dette er at når new direction kalles etter at bestillingen er lagt til i køen, vil switchen i new direction-funksjonen ikke gi utslag fordi heisen er i bevegelsestilstand. Heisen vil fullføre bestillingen i 3. etasje, kalle på new direction etter tre sekunder, og sette kurs mot 1. etasje.

I det andre scenariet oppstår trøbbel. Grunnen til dette er at når den nye bestillingen kommer, vil foreslått løsning kalle new-direction og heisen går mot 1. etasje før 3 sekunder er over. Man kan selvsagt legge til støtte-logikk som sørger for riktig oppførsel, men sannsynligheten for at dette er eneste støtte-logikk som må legges til for å behandle alle scenarier en svært liten. Alternativt må man tenke nøye gjennom ved hvilke tidspunkt man faktisk tillater funksjonskall av new direction, men da begynner løsningen uansett å ligne på designet valgt i dette prosjektet. I lys av dette argumenteres det at designet valgt i dette prosjektet kan beholdes.

Dersom løsningen skulle blitt skalert, kunne man vurdert å skille noen av funskjonene i Controller ut i et støttebibliotek. Dette gjelder eksempelvis funksjonene controllerElevatorAtFloor og controllerRequestBetween (se Doxygen-dokumentasjon for detaljer), som ikke er en direkte del av heisstyringen, men som brukes av funksjonene som står for styringen. Da kunne flere eventuelle moduler brukt støttebiblioteket uten å være avhengig av Controller-modulen, som gir færre avhengigheter og dermed høyere vedlikeholdsgrad.

6 Appendiks

6.1 Kravspesifikasjon

A Appendiks - Heisspesifkasjoner

A.1 Oppstart

Punkt	Beskrivelse
O1	Ved oppstart skal heisen alltid komme til en definert tilstand. En definert tilstand betyr at styresystemet vet hvilken etasje heisen står i.
O2	Om heisen starter i en udefinert tilstand, skal heissystemet ignorere alle forsøk på å gjøre bestillinger, før systemet er kommet i en definert tilstand.
O3	Heissystemet skal ikke ta i betraktning urealistiske start-betingelser, som at heisen er over fjerde etasje, eller under første etasje idet systemet skrus på.

A.2 Håndtering av bestillinger

Punkt	Beskrivelse
H1	Det skal ikke være mulig å komme i en situasjon hvor en bestilling ikke blir tatt. Alle bestillinger skal betjenes selv om nye bestillinger opprettes.
H2	Heisen skal ikke betjene bestillinger fra utenfor heisrommet om heisen er i bevegelse i motsatt retning av bestillingen.
H3	Når heisen først stopper i en etasje, skal det antas at alle som venter i etasjen går på, og at alle som skal av i etasjen går av. Dermed skal alle ordre i etasjen være regnet som ekspedert.
H4	Heisen skal stå stille om den ikke har noen ubetjente bestillinger.

A.3 Bestillingslys- og etasjelys

Punkt	Beskrivelse
L1	Når en bestilling gjøres, skal lyset i bestillingsknappen lyse helt til bestillingen er utført. Dette gjelder både bestillinger inne i heisen, og bestillinger utenfor.
L2	Om en bestillingsknapp ikke har en tilhørende bestilling, skal lyset i knappen være slukket.
L3	Når heisen er i en etasje skal korrekt etasjelys være tent.
L4	Når heisen er i bevegelse mellom to etasjer, skal etasjelyset til etasjen heisen sist var i være tent.
L5	Kun ett etasjelys skal være tent av gangen.
L6	Stoppknappen skal lyse så lenge denne er trykket inne. Den skal slukkes straks knappen slippes.

A.4 Heis-dør

Punkt	Beskrivelse
D1	Når heisen ankommer en etasje det er gjort bestilling til, skal døren åpnes i 3 sekunder, for deretter å lukkes.
D2	Heisen skal være lukket når den ikke har ubetjente bestillinger.
D3	Hvis stoppknappen trykkes mens heisen er i en etasje, skal døren åpne seg. Døren skal forholde seg åpen så lenge stoppknappen er aktivert, og ytterligere 3 sekunder etter at stoppknappen er sluppet. Deretter skal døren lukke seg.
D4	Om obstruksjonsbryteren er aktivert mens døren først er åpen, skal den forbli åpen så lenge bryteren er aktiv. Når obstruksjonssignalet går lavt, skal døren lukke seg etter 3 sekunder.

A.5 Sikkerhet

Punkt	Beskrivelse
S1	Heisen skal alltid stå stille når døren er åpen.
S2	Heisdøren skal aldri åpne seg utenfor en etasje.
S3	Heisen skal aldri kjøre utenfor området definert av første til fjerde etasje.
S4	Om stoppknappen trykkes, skal heisen stoppe momentant.
S5	Om stoppknappen trykkes, skal alle heisens ubetjente bestillinger slettes.
S6	Så lenge stoppknappen holdes inne, skal heisen ignorere alle forsøk på å gjøre bestillinger.
S7	Etter at stoppknappen er blitt sluppet, skal heisen stå i ro til den får nye bestillinger.

A.6 Robusthet

Punkt	Beskrivelse
R1	Obstruksjonsbryteren skal ikke påvirke systemet når døren ikke er åpen.
R2	Det skal ikke være nødvendig å starte programmet på nytt som følge av eksempelvis udefinert oppførsel som for eksempel at programmet krasjer, eller minnelekkasje.
R3	Etter at heisen først er kommet i en definert tilstand ved oppstart, skal ikke heisen trenge flere kalibreringsrunder for å vite hvor den er.

Figur 6: Kravspesifikasjon