Erhvervsakademi MidtVest

Flextursprojekt

Vejledere: Hans Iversen, Anders Petersen, Flemming Jensen

Martin Zylauv Kristensen, Steffen Hagelskjær

03-06-2016

Omfang: 37,7 normalsider/82983 tegn

Indholdsfortegnelse

Indledning

Dette førsteårs projekt er udarbejdet af Martin Zylauv Kristensen og Steffen Hagelskjær i maj 2016. Projektets emne er udvikling af et IT-system, Flextur, til MidtTrafik. I dette projekt har vi udarbejdet en IT-forundersøgelse som opstart af projektet for at derefter kunne udforme vores projekt i form af systemudviklingsartefakter samt konstruktionen af et eksekverbart program. Der vil først snakkes om selve problemdomænet for projektet hvor vi kommer omkring problemet stillet for vores opgave og krav, for derefter at gå videre til metodevalg for udarbejdelse af projektet. Så kommer der en procesbeskrivelse, hvor vi kommer omkring vores udarbejdelse af hele projektet ved et skema og forklaring til de enkelte punkter. En produktbeskrivelse med beskrivelse af alle systemudviklingsartefakter samt kode bliver gået i dybden på næstsidst, og til sidst en konklusion der dækker vores endelige resultat ud fra vores problemformulering og krav sat til opgaven.

Problemformulering

MidtTrafiks nuværende hjemmeside og applikation dækker ikke de behov visse segmenter af deres brugere har behov for, for nemt at kunne bestille en kørsel via Flextrafik. Normalt er de brugere der har behov for en speciel kørsel på et specielt tidspunkt folk der har visse specielle behov som ikke normalt er dækket af bus- eller taxamuligheder. Tit har disse brugere enten ikke smartphone eller forståelse nok for teknologi til at kunne bestille uden behov for ekstra hjælp.

For at imødekomme problemet som dette segment har, vil vi udvikle en løsning til MidtTrafik i form af et bestillings- og oversigtsystem for MidtTrafiks kunder og personale til konceptet Flextrafik. Denne løsning skal ende med at gøre det nemmere for brugerne med visse behov at bestille en kørsel via MidtTrafiks Flextrafik.

Undervejs i denne rapport vil vi forsøge at imødekomme flere forskellige problemstillinger, såsom hvilke krav der ligger indenfor vores projekt, vi vil se om vi kan imødekomme disse krav og om vi kan lave et program der er tilstrækkeligt godt og nemt at anvende til at opfylde brugerens behov, og samtidig dække MidtTrafiks ønske til et professionelt IT-program.

Afgrænsning

Vi vil i dette projekt kort komme rundt omkring alle artefakter der er relevante for det endelige produkt. Visse artefakter vil blive udeladt og andre vil blive forklaret kort, da vi gerne vil bredt komme omkring alt.

Dette projekt vil blive udelukkende udviklet i Java med udviklingsværktøjet Eclipse og databaseværktøjet HSQLDB. Det primære fokus ligger på at opfylde de krav vi har i kravspecifikationerne og visionsdokumentet, så eventuelle nice-to-haves kan være udeladt.

Problemdomæne

Til beskrivelse af vores problemdomæne har vi valgt at bruge artefakter udarbejdet fra MUST-metoden for at give os en god start på vores forundersøgelse. I denne must-metode har vi valgt følgende metoder til at beskrive det:

- Udarbejdelse af et visionsdokument
- SWOT analyse af MidtTrafik
- Dokumentanalyse af opgaveformuleringen
- Eksperimenter med prototyper (De første mockups)

Vi har bevidst fravalgt de andre artefakter i MUST-metodens i alt 17 teknikker for vi føler at vi kommer frem til en god start på løsningsforslaget med disse teknikker valgt fra MUST. Vi kunne have valgt en funktionsanalyse af virksomheden, men vi valgte at lave en SWOT-analyse i stedet, da den giver et mere overordnet blik af virksomhedens strategiske situation, hvilket vi vil tage udgangspunkt i når vi vil udvikle vores løsning til MidtTrafik. I SWOT analysen inddrager vi også virksomhedens konkurrencesituation, hvilket vi vil ind og kigge på, og dermed prøve at forbedre denne med vores løsning, og om den passer ind i de svagheder som virksomheden end måtte have. Da vi også har konkrete krav til programmet beskrevet i opgaveformuleringen, samt analyseret dette i dokumentanalysen, har vi fravalgt den tidskrævende teknik observation, hvor man f.eks. kan bruge en konkurrerende applikation eller lignende applikation, og observere denne, hvilket kunne have givet gode ideer til vores løsning. Men som nævnt at vi allerede fra start havde en lang række krav til programmet fravalgte vi denne, da vi ville fokusere på opfyldningen af disse krav, og ikke opfyldningen af en masse nice-to-haves vi eventuelt kunne have sat som krav ved denne observation.

MUST bliver dog også anvendt senere i projektet, hvor der er andre teknikker anvendt. Såsom reviews til kvalitetssikring og en referencelinje planlægning til projektstyringen.

Visionsdokument(Steffen)

Hvordan kom vi frem til visionsdokumentet

Vi kom frem til vores visionsdokument ved at kigge delvist på MidtTrafiks egen hjemmeside, og delvist ved brug af vores opgavebeskrivelse. Vores visionstekst kommer kort og godt omkring de ting vi forestiller os vi kan nå frem til med projektet, på en måde der fuldfører de krav vi har fået stillet. Derefter lavede vi vores interessantanalyse hvor vi har gået mere i dybden og tænkt over alle de forskellige interessenter der kunne være involveret i vores opgaves endelige produkt, og har prøvet så godt som muligt at formidle hvordan vi forventer det har effekt på alle interessenter. Til sidst har vi så lavet en featureliste, som ud fra visionsteksten, interessentanalysen og vores projektbeskrivelse går ind på de ting som der ligger krav for hvad vi skal lave i projektet samt de ting vi tidligere har overvejet, for at implementere det på en måde vi ser mest effektivt og optimalt til det endelige resultat.

Hvad den betyder for vores projekt

Visionsdokumentet betyder meget for projektet, da det endelige produkt skal kunne leve op til vores vision. Visionsdokumentet baner vej for at vi kan komme ordentligt i gang med projektet på en overskuelig måde, og er derfor en slags guide til den basale, overordnede retning vi styrer opgaven i, men er udover det også det endelige mål for vores projekt, da visionen er et succeskrav.

SWOT Analyse

Strengths

En styrke for MidtTrafik indenfor deres flextur er det yngre segment. De har en app på markedet der gør det nemt at bestille, og alle der har forstand på at navigere en smartphone kan derved bestille en tur med deres flexturs system. Udover det har de også mulighed på nuværende tidspunkt at bestille online på deres hjemmeside, som på trods af at være vores opgave at gøre bedre, giver det en mulighed for det segment der ingen smartphone ejer eller ikke har forstand på at bestille igennem det. Derfor er det stadig en styrke at kunne give mulighed for det online. Der er desuden en sidste mulighed for at bestille, hvor man kan ringe ind til MidtTrafik og verbalt bestille en tur, som er en god styrke at have hvis folk ikke kan finde ud af noget som helst med teknologi.

Weaknesses

En svaghed for MidtTrafik ved deres flextur er, samtidig at være en styrke, deres app. Hvis man kigger på anmeldelser ligger deres app kun på 3,1 stjerne ud af 5 på Google's appstore ud fra 31 stemmer, hvori mange klager over deres mangel af bedre betalingsmuligheder og småfejl i systemet. Dog er den største svaghed for MidtTrafiks flextur naturligvis deres mangel på en bedre bestillingsform online. Deres

nuværende er alt for besværlig at anvende for folk der ikke har mange færdigheder inden for computere, og gør det derfor besværligt at gå igennem en bestilling. Dette gør højst sandsynligt at mange folk er nødsaget til at ringe ind i stedet for at bestille online, som gør at brugeren går igennem en del mere arbejde end der ønskes for at de kan nå frem til deres endelige mål, som jo naturligvis ikke er ønsket fra hverken brugerens eller MidtTrafiks side.

Opportunities

En mulighed der ligger for MidtTrafik er selvfølgelig generelt at forbedre deres muligheder. Det er en stor mulighed at kunne forbedre deres app, så måske at selv folk der ikke har så meget forstand på smartphone telefoner ville kunne finde ud af at navigere i selve programmet når det er hentet ned, eller bare generelt lytte til de fejlmeldinger og problemer brugere har haft med dette app. Dog er den største mulighed at udvide omkring det ældre segment med hensyn til online bestilling. Eftersom det nuværende er en kæmpe svaghed at det ældre segment har alt for svært ved at bestille en tur med deres flextur, er det en rigtigt stor mulighed at kunne sørge for at dette måske lidt ældre segment får gjort det nemt og forståeligt at køre igennem, da dette vil give en god basis for at give brugeren meget mere kontrol over hvor let det er at bestille en tur, og aldrig mere bør være nødsaget eller tvunget til at ringe ind til MidtTrafik eller (formodet) spørge venner/familie om hjælp med hensyn til at booke. Så generelt at udvide i deres program til at være meget mere brugervenligt vil uden tvivl give en fordel i fremtiden for MidtTrafik og deres brugere.

Threats

En fare ved udvidelse af deres systemer for MidtTrafik kommer den trussel også at deres programmer, på trods af at blive gjort mere brugervenlige og nemmere at styre, stadig ikke hjælper brugeren nok i sidste ende til at gøre en forskel fra det nuværende. Det er altid besværligt præcist at vide hvad ens bruger kan gøre, og brugertests er generelt en god måde at komme igennem det på, men det garanterer ikke noget for et helt segment. Det er dog en mindre fare ved udvidelse af deres nuværende system. Et stort problem er at hvis deres nye system er udvidet til et unødvendigt punkt og skubber alt for meget på brugeren på trods af at være nemmere at anvende. Dette kan muligvis resultere i tvivl med hensyn til bestilling for brugeren, og derved også miste brugervenlighed samtidig, på trods af intention var at skærpe brugervenligheden.

Hvad betyder SWOT-analysen for projektet

SWOT-analysen kortlægger i bund og grund virksomhedens strategiske- og konkurrencesituation[1], samtidig med at give overblik over risikofaktorer. SWOT-analysen er derfor god til at give indblik i virksomhedens muligheder og ideologier, og gør det muligt at arbejde ud fra den viden det giver. Det er nemlig nødvendigt

[1] [Keld Bødker, Finn Kensing, Jesper Simonsen] Professionel it-forundersøgelse s.233

at tænkte over hvilket firma man arbejder for, da der er stor forskel på hvad de forventer af det endelige produkt.

Intuitivt GUI

Det er også meget nødvendigt at tænke over den grafiske brugergrænseflade man kommer til at anvende ud fra SWOT-analyse. Det er noget vi gerne vil have sat som krav på trods af det ikke er sat i forvejen, altså at bruge SWOT-analysen til at vide hvordan vi skal designe en brugergrænseflade der fungerer indenfor det segment og de kvaliteter vi har med at gøre.

Step-by-step

Ud fra SWOT-analysen kan vi også komme ind og kigge på step-by-step, som dikterer hvordan en bruger anvender programmet, og hvilken rækkefølge det forventes at de gør ting. Step-by-step er derfor en brugbar måde at optimere oplevelsen for brugeren, så det bliver mere intuitivt at anvende programmet, og gør sig også mærke i hvad der indgår i SWOT-analysen.

Dokumentanalyse af opgaveformuleringen(Martin)

For at få et bedre overblik over den stillede opgave valgte vi at lave en mindre dokumentanalyse af den udleverede opgaveformulering. Vi gjorde dette for at få styr på alle de krav som projektet skulle kunne stille op til, og for at sikre os at projektet nemlig i sidste ende ville kunne leve op til disse, uden at vi skulle lave større ændringer i projektstrukturen i sidste øjeblik. Notatarket for denne dokumentanalyse kan findes i bilag 1, og viser vores notater som vi lavede fælles, hvor vi havde opgaveformuleringen på en tavle projekteret med en projektor fra vores computer. Med denne analyse af dokumentet er vi sikret at vores antagelser af dokumentet er skrevet ned allerede i forundersøgelse, så der ikke sker misforståelse for projektet imens det er igangværende[2].

Kravspecifikationer(Martin)

Baseret på vores visionsdokument, SWOT analyse og dokumentanalyse af opgaveformuleringen er vi kommet frem til visse krav som vores program skal kunne overholde i sin færdige version. Disse kravspecifikationer sikrer kunden et billede af hvad programmet som minimum skal kunne præstere når det bliver afleveret i sin endelige form, samt hvad dette program skal kunne i forhold til andre tilbud fra andre udviklere som MidtTrafik end måtte have fået. Der er jo normalt en form for tilbudsproces, hvor de

[2] [Keld Bødker, Finn Kensing, Jesper Simonsen] Professionel it-forundersøgelse s.232

ser på hvad udviklerne kan tilbyde, og til hvilken pris. Vi har valgt at inddele vores kravspecifikationer ind i 3 overordnede punkter, Funktionelle krav, Ikke-funktionelle krav og projektbegrænsninger. Det giver nemlig det bedste overblik for udviklingsteamet, samt for kunden der hurtigt skal have overblik over hvilket et udviklerteam de vil vælge til deres opgave[3].

Funktionelle krav

Systemet skal kunne:

- Vise en brugerprofil.
- Håndtere redigering af brugerprofil.
- Vise brugerens historik af bestilte kørsler.
- Håndtere kommentering af kørsler under bestilling.
- Håndtere og gemme godkendelse af kørsler.
- Håndtere tildeling af biler til kørsler.
- Håndtere bestilling af kørsel fra administrator siden.
- Håndtere kommentering af kørsler fra administrator siden.
- Håndtere visning af afholdte ture (ture der er tildelt en bil og som er før den pågældende dato)
- Håndtere eksportering til csv fil, både fra brugeren og administratoren.
- Håndtere login for bruger og administrator
- Håndtere visning af forskellige skærmbilleder i forhold til om man er bruger eller administrator.
- Gemme oplysninger som programmet håndterer i en lokal database på brugerens computer.

Disse funktionelle krav er med til at kvalitetssikre vores udviklingsproces, samt at give os en stor målrettet slut-milepæl som vi minimum skal sigte efter at vores program skal kunne udføre. Udover disse funktionelle krav er der en række ikke-funktionelle krav som vi har i vores program.

Ikke-funktionelle krav

Vores ikke-funktionelle krav er delt ind i 2 underpunkter for at skabe det bedste overblik over disse for kunden. Nemlig performancebegrænsninger, og projektbegrænsninger. Performance begrænsninger omhandler mest hvordan at programmet skal præstere ude hos kunden når programmet er færdiglavet, og projektbegrænsninger omhandler begrænsninger inden for deadlines og ressourcer.

Performancebegrænsninger

- Brugergrænsefladen skal være letforståelig og intuitiv.
- Der skal være en hurtig feedback på handlinger i brugergrænsefladen i forhold til brugerens oplevelse af ventetider.

[3] http://www.info-ark.dk/article.php?alias=kravspec_2

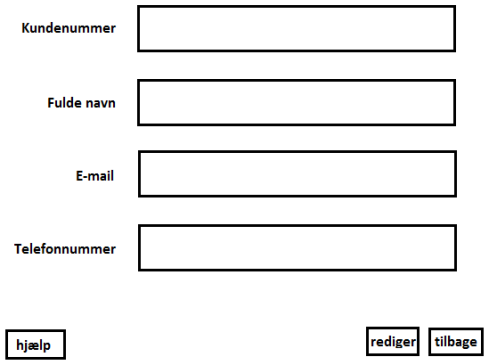
- Brugergrænsefladen skal kunne håndtere eventuelle fejl i systemet, og vise dem på en letforståelig måde som den normale bruger kan relatere til, og eventuelt skyde til handling med, såsom at kontakte kundeservice eller genstarte applikationen.

Projektbegrænsninger

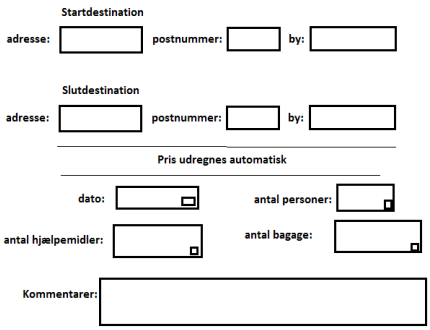
- Deadline på en færdig applikation samt systemudviklingsdokumentation er d. 3/6/2016
- 2 udviklere er til rådighed til at udvikle applikationen samt at udarbejde dokumentation til applikationen

Eksperimenter med mockups/tænke højt test(Martin)

Vi har valgt at lave prototyper bestående af mockups af de første par skærmbilleder til vores program, som er baseret på kravspecifikationerne samt visionsdokumentet og de andre artefakter som vi har udarbejdet. Skærmbillederne kan ses på figur 1 og figur 2, samt i fuld version i bilag 2 bilag 3. Vi lavede en tænke-højt-test mens vi udarbejdede disse grove mockups, hvor vi sad og beskrev som om at vi brugte programmet, og gerne ville bestille en kørsel/redigere profilen. Notaterne til tænke-højt - testen kan ses i bilag 4 og de er en sammenfatning af vores oplevelser af vores eksperimenter med disse prototyper. Vi kom bl.a. frem til at vi gerne ville have at prisen kunne udregnes imens man indtastede de andre informationer som var påkrævede for at bestille turen, men som ikke var påkrævede for at få et pris tilbud udregnet. Vi kom også frem til en cirka-placering som vi gerne ville have at vores mere fint-udarbejdede mockup-objekter skulle være placeret henne i skærmbilledet. Udover dette kom vi frem til at målgruppen jo ikke går op i flot grafik, da vi antager at de er en del af det ældre segment hvis de ikke ejer en smartphone, derfor skal der bare være enkelt brugergrænseflade som der let kan tilgås, og hvor der ikke er meget grafik eller skjulte menuer som f.eks. at hvis man højre klikker, så kommer der skjulte funktioner frem.



Figur 2 skærmbillede 2 af læs/rediger profil



Figur 1 bestil kørsel mockup

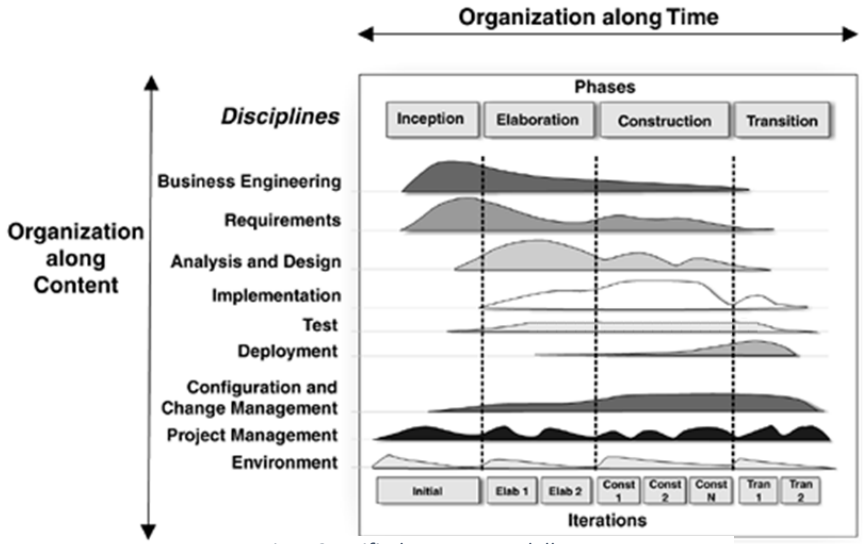
Metodevalg

Vi har i vores projekt valgt at arbejde med Rational Unified Process, hvilket er en proces som vi har godt kendskab med, og som vi tidligere har haft gode erfaringer med. I dette afsnit vil der forklares om hvorfor vi har valgt denne, hvad der specielt er godt ved den, og hvorfor vi har valgt andre metoder til projektstyring og udvikling fra.

Hvorfor har vi valgt Rational Unified Process? (Martin)

Vi valgt den som sagt grundet vores tidligere erfaringer med den, og fordi den har visse gode egenskaber knyttet til sig som har vist sig at hjælpe til med kvalitetssikringen for selve udviklingsprocessen, så slutbrugeren står med et produkt der har fulgt tidsplanen samt det angivede budget der var fra starten af[4]. Rational Unified Process (RUP) er blevet udviklet af IBM, som jævnligt opdaterer og vedligeholder denne udviklingsmetode, hvilket gør den til en af de bedste udviklingsmetoder der p.t. er til de behov som vi som udviklere har. Grundet RUP, kunne vi også "nøjes" med at offentliggøre vores produkt tidligt med manglende funktionaliteter, hvis de risici som vi definererede i risikoanalysen[5] skulle gå hen og blive en realitet. [6] Udover dette har RUP mange andre gode kvaliteter, såsom muligheden for bedre kommunikation i udviklingsteamet, bedre kontrol over komplekse projekter, samt et mindre omkostningsrigt projekt, hvilket igen, gør det til en kvalitetssikring hos kunden i den form at projektet er gået som planlagt, og til den aftalte pris.

RUP følger den iterative form for udvikling, hvor den i starten har en iterativ evolutionær udvikling, hvor kravene er usikre, og selve udviklingen af produktet afdækker de krav som produktet skal have, f.eks. vha. Prototyping m. Mockups som vi viste i problemdomænebeskrivelse. Senere i projektet bliver det dog til en inkrementel udvikling, hvor vi har fået specificeret kravene, og selve udviklingen planlægges efter de prioriteringer som disse krav end måtte have[7]. Ved hjælp



Figur 3 Unified Process Modellen
Kilde: [Philippe Krutchen] The Rational Unified Process An Introduction s.22

[4] [Philippe Krutchen] The Rational Unified Process An Introduction s. 17
[5] Se bilag 13 for vores risikoanalyse
[6] [Philippe Krutchen] The Rational Unified Process An Introduction s. 24
[7] [PowerPoint af Anders Petersen] UP-01B Procesmodeller

af denne iterative udvikling kan vi dermed fremstå med et delprodukt ved hver iteration, som der i realiteten kan gives til kunden for at vise fremskridt af produktet. Dette kan være udvikling af en enkelt usecase der er færdig frem til implementeringen, så de kan få en forsmag på programmet. Det er det som vi har prioriteret meget højt i vores proces i dette projekt, at vi kunne fremvise et færdigt produkt ved hver iteration, så hvis der skulle opstå nogle situationer der var forbundet med vores risikoanalyse, så ville vi have et delprodukt, som vi kunne fremvise og aflevere som halvfærdigt produkt, som der så kunne arbejdes videre på når disse risikofyldte situationer var overstået.

På figur 3 kan Unified Process ses i form af hvilke arbejdsbyrder der burde ligges vægt på i hvilke iterationer. Vi har stort set fulgt denne i vores udviklingsproces med undtagelse af mindre ændringer, som har været til hvis vi f.eks. har været tæt på at færdiggøre en enkelt implementation af en usecase, eller hvis vi skulle lave noget om midt i projektet. Sådan nogle processer er jo til at blive tilpasset, og det er også det vi har brugt den til, nemlig at tilpasse den så den passede til netop vores projekt, hvor kravene var indsamlet fra starten af, og der ikke var det store behov for en evolutionær iterations udvikling i starten, bortset fra et par enkelte prototype tests med mockups.

Planlægningen af den første iteration(Martin)

Vi planlagde vores første iteration som noget af det første da vi påbegyndte projektet. Vi startede ud med at planlægge fremstilingen af de første artefakter som vores første milepæl påkrævede os at have færdigt en uge efter, samt fremstillingen af de første artefakter til vores første og vigtigste usecase, nemlig usecase 1. Milepælen for denne iteration ville have været at have en fuldt ud virkende DB-prototype, da det var en risiko vi havde påregnet i vores risikoanalyse at den ville være svær at lave grundet vores mangelende kendskab i teamet, og at vi havde fuldt fungerende mockups vi kunne fremvise for de første og vigtigste par usecases. Alt dette foregik efter planen, men da milepæls fristen blev skubbet fremad, så påbegyndte vi bare anden iteration så vi var på forkant med projektplanen.

Fravalg af andre metoder(Steffen)

Vandfaldsmodellen

Vandfaldsmodellen er en model der går ud fra at man arbejder alle artefakter af denne samme slags fuldkommen igennem før man går videre, det vil f.eks. sige at man lave alle uformelle usecases for projektet, derefter alle formelle uscases, alle operationskontrakter osv. Vi har valgt at vælge vandfaldsmodellen fra, det vi på forhånd vidste hvad vores projekts omfang var, og havde derfor ikke et behov for at lave alt forebyggende før vi gik i gang med koden. Et stort problem ved vandfaldsmodellen i et projekt af vores omfang er også at hvis vi endte med at skulle lave noget om i kodningen til allersidst, f.eks.

tilføje en ekstra knap til et af vores GUIs, ville det have taget rigtigt lang tid at komme omkring kravændringerne i vores dokumenter. Så for at vandfaldsmodellen ville have været brugbar for os, skulle vi udover de fordele der ligger i Unified Process have været ekstra opmærksomme på de fejl der kunne komme senere i projektet for at undgå at vi ville miste ekstrem meget ekstra arbejdstid.

Agile udviklingsmetoder

En agil udviklingsmetode er essentielt en fremgangsmåde for sit projekt hvor man prioriterer kunden eller virksomheden frem for alt. Det vil sige, at man ikke går så meget op i dokumentation og det forebyggende arbejde som rent faktisk at få et endeligt produkt der matcher det man havde fået stillet til opgave i det hele taget. Dette er en fordel med hensyn til at få tingene lavet så billigt som muligt, og ville uden tvivl have fungeret for vores projekt med hensyn til at få lavet et færdigt produkt, men det nedprioriterer dokumentation som betyder at hvis man støder ind i problemer, er det svært at konkretisere og få overskud over de problemer der ligger og hvordan man præcist løser dem. Dette gør også at teoretisk når man er færdig med at udvikle sit projekt, hvis man senere skal arbejde på det igen og vedligeholde det er det svært at sætte sig ind i hvordan hele programmet konkret er sat op. Så for vores projekt har vi valgt de fra, eftersom det ikke er særligt struktureret, og ikke godt for fremtiden, specielt pga. andre udviklere skal bruge det vi har lavet til at sammenkoble det på en hjemmeside. Den mest anvendte form for agil udvikling er SCRUM, som er en iterationsmæssig fremgangsmåde for agil udvikling, hvor man i faser prioriterer hvad der er vigtigst at få lavet færdigt for at nå en deadline, men det er heller ikke relevant for os, eftersom vi har haft en god konkretiseret opgave hvor vi havde et godt overblik over hvor længe det ville taget at få alt lavet færdigt med alt forebyggende dokumentation.

Spiralmodellen

Spiralmodellen er en model der går i ring i 4 forskellige faser. Planlægningsfasen, Risikoanalyse, ingeniørfasen og evalueringsfasen. Modellen er meget god at anvende til projekter hvor man har meget at arbejde igennem, da den tager meget hensyn til eventuelle risikoer løbende og giver plads til meget omtanke for projektet. Det er også derfor ikke så vigtigt for os at anvende i dette projekt, da der ikke ligger voldsomt mange problemer i fejl hen af vejen, eftersom det ikke er et særligt stort system. Spiralmodellen gør sig meget mere gældende for meget større projekter med mange personer der arbejder på det på samme tid, hvor det er nødvendigt at tage stikprøver og evaluere sit arbejde hver gang man har taget et skridt fremad.

Procesbeskrivelse

Iterations/faseplanen(Steffen)

Grunden til vi valgte Microsoft Project til at føre en plan over vores projekt er, at det er et program der er nemt at sætte et overskueligt skema op, og få strukturingen til at passe med da der er lidt mere åben plads til tænkning. Når vi arbejder i Project med en oversigt over de forskellige iterationer er det også samtidig nemt at markere om man er fuldkomment færdig med en opgave, eller om der mangler noget for at få det færdiggjort. Eksempelvis hvis vi var usikre på om et systemsekvensdiagram var lavet korrekt kunne vi blot markere den specifikke del i planen som gul frem for grøn, for at vise den er påbegyndt og måske til en hvis grad færdig, dog ikke helt fuldent, og kan på den måde skabe et hurtigt overblik over hvilke artefakter vi mangler at få arbejdet på. Udover det gør Microsoft Project det også nemt at overskue en fremgang i projektet, da den viser data over hvor mange timer der er sat og, samt en masse kørende linjer der viser hvor meget det fylder i forhold til tid i det fuldkomne projekt. Hele vores projektplan kan findes under bilag 5 i form af skærmbilleder taget fra den. Alle artefakter der ikke direkte er refereret til i form af bilag nummer kan findes i bilag 14, hvor man kan se alle de udarbejdede artefakter fra dette projekt.

Hvorfor ikke referencelinjeplan?

Referencelinjeplanen er en opsætning af projektets plan hvor man sætter specifikke punkter der skal arbejdes på i henhold til specifikke tidspunkter. Den er meget bedre til at skabe et præcist overblik over hvad man skal nå, og giver derfor en meget struktureret måde at arbejde sig frem med planen. Dog valgte vi den fra, da på trods af at dette er en fordel når det kommer til at få et godt overblik over de specifikke punkter, giver det ikke et særligt godt overblik i forhold til balance af punkter. Den giver ikke frihed til at komme tilbage og arbejde videre på ting, i hvert fald ikke uden at det bliver alt for uoverskueligt når der er så mange forskellige dokumenter osv. der er smidt på samme række. Det kan nemlig også rent visuelt blive for meget at overskue, hvis projektet har en masse forskellige punkter, og er derfor enten meget præcist og mister uoverskuelighed, eller mere generelt og derved mister det meget af det der gør det en god plan at følge i det hele taget.

Udvikling af faseplan

Vores udvikling af faseplanen startede ud fra teorien bag Unified Process, hvor man starter med det forebyggende arbejde og indblik i de krav vi har fået stillet for de forskellige iterationer. Det ville sige at den første iteration er sat op efter at den første iteration har visse krav, som f.eks. vores visionsdokument samt en prototype af database, og så har vi også inkluderet arbejde på dokumenter til UC1 samt interfaces for at komme i gang. Til sidst i første iterationsplan begynder vi også på opsætningen af anden iteration for at

være klar på forhånd og ikke afsætte tid til at lave anden iterationsplanen under den anden iteration. Under anden iteration er der planlagt ud med mere elaboration, hvor vi går mere i dybden med dokumenter til forarbejde for programmeringen, altså hvor vi strukturerede UC 2 til 6 og lidt løbende programmering på de første 3 usecases med interfaces og lidt andet. Her bliver iterationsplanen for iteration 3 også sat op så vi var klar på at arbejde videre. Tredje iteration er så hvor der bliver gået meget mere i dybden med programmering og aktuelt arbejde på projektet, fordi meget af baggrundsarbejdet er overstået og det er lagt klar til man kan komme i gang. Denne iteration går derfor mere i gang med programmering, men sætter selvfølgelig også tid af til lidt af det andet baggrundsarbejde, da ikke alle usecases er gennemgået endnu. Fjerde iteration er den sidste iteration hvor programmering bliver gennemgået, hvor alt bliver lavet færdigt, og tingene bliver kigget igennem for at sikre os at der ingen fejl er i vores dokumenter i forhold til det aktuelle program, så det er nemt at finde rundt i vores program. Desuden ligger brugertest og generelle tests i programmet meget fremtrædende under denne iteration, da programmet begynder at være fuldendt og vi skal sikre os at der ikke er nogle problemer med programmet og at brugerne i sidste ende rent faktisk kan anvende det endelige program. Femte iteration er det sidste iteration, og er meget lille. I femte iteration er det sådan set bare eksportering af vores .jar fil og fuldende rapport og den endelige aflevering.

Inception (Steffen)

1. Iteration

Identificering af usecases

I vores første iteration begyndte vi med at identificere vores usecases. Her satte vi aktører op i et usecase diagram der ville være relevante at have usecases ud fra, og ud fra vores opgavebeskrivelse beskrev vi de overordnede usecases så vi havde dem identificeret og klar til at lave senere i iterationen samt de kommende iterationer.

Beskriv Usecase 1-3 formelt

Her begyndte vi at beskrive vores første 3 usecases formelt for at have noget at arbejde konkret ud fra. Vi sad og arbejde sammen om udarbejdelse af de formelle usecases for at sikre at vi begge er enige om fremgangsmåden på programmet og hvad de forskellige usecases præcist skal indeholde i forhold til programmet. Grunden til vi udarbejdede alle 3 på samme tid på trods af det ikke er den normale fremgangsmåde i UP, er for at vi havde dem klar til at arbejde videre, så vi ikke skulle sidde sammen og udarbejde en usecase senere.

Lav visionsdokument

Her sad vi også og udarbejdede dokumenterne sammen. Grunden til at visionsdokumentet er vigtigt at komme sammen om at lave, er fordi man skal være helt sikker på hvad opgaven går ud på og hvilke krav vi vil gennemføre i projektet. Visionsdokumentet skal dække det endelige resultat så tæt som muligt, da det ikke bare er en generel idé, men nærmere det forventede resultat til allersidst for programmet.

Mockup for UC 1-3

Vores måde at lave mockups på var hurtigt at snakke om hvordan vi cirka ville have vores program til at se ud så det ville være nemt at arbejde ud for brugeren eller MidtTrafik, alt efter hvem der er aktøren i den aktuelle usecase. Den måde vi lavede dem på var med en hurtig overordnet, meget sjusket skitse i Microsoft Paint, hvorefter vi arbejdede den ud i SceneBuilder. Her kom det så tæt på det endelige resultat vi kunne komme inden vi begynder med den aktuelle programmering og finder ud af hvordan realiteten kommer til at passe med mockuppet, og samtidig giver det mere en slags prototype for vores program end blot at være en tegning, og kan dermed være nemmere at anvende til brugertests senere.

Lav domænemodel

Ud fra vores UC 1-3 formelle usecases kunne vi udarbejde en domænemodel, som viser sammenhængen mellem de forskellige begreber vi kommer til at have i vores program ud fra vores usecases. Når man har en færdig domænemodel er det også meget nemt at sætte en database op ud fra det, som er brugbart eftersom vi skulle lave en database prototype senere i den første iteration.

Lav artefakter for UC1

Vi startede her at lave artefakterne for Usecase 1, så vi ville være klar til at lave interfaces til den senere i iterationen, og bagefter være klar til at udarbejde og implementere den. Vi startede med aktivitetsdiagrammet, som bedre viser hvert skridt i programmet, og hvilke dataformer der er i spil udover blot hvad programmet gør. Ud fra aktivitetsdiagrammet lavede vi systemsekvensdiagram, som viser hvordan usecasen bliver anvendt i programmet uden at vise de programmeringsmæssige detaljer, for at visualisere kommunikation med systemet, så man kan se hvordan systemet vil respondere til de forskellige skridt. Desuden identificeres systemoperationer i systemsekvensdiagrammet. Det næste vi lavede var operationskontraker, hvor vi kort beskriver detaljer omkring vores systemoperationer med tekstbeskrivelser sat op efter en skabelon. Der er sat fokus på hvad der sker i operationskontrakten, ikke hvordan det sker. Efterfølgende lavede vi DCD og sekvensdiagram omtrent samtidig, og i sammenhæng med sekvensdiagrammet og operationskontrakten kom frem til hvad der skulle stå i klasserne i DCD. Vi

valgte at lave vores klasser mere som interfaces for at benytte den evolutionære iterative fremgang, som man plejer at gøre i starten ifølge UP da kravene er lidt løsere, og vi ved endnu ikke helt præcist hvordan programmet skal laves, så vi kan prøve os mere frem til et resultat. Sekvensdiagrammet blev udarbejdet mere fra operationskontrakten hvor den viser hvordan tingene sker i programmet og viser hvordan det skal sættes op programmeringsmæssigt.

Lav iterationsplan for iteration 2

Nu begyndte vi så på iterationsplanen for anden iteration, elaboration-fasen, så vi ville være klar på forhånd til at arbejde videre med vores projekt så snart denne iteration var overstået. Her planlagde vi så de ting der hører ind under elaborationsfasen og andre ting vi havde behov for at gå videre med for at programmet kunne laves videre.

DB prototype

En del af milepæl A var at lave en prototype af vores database. Vi ville egentlig have gået i gang med den senere i projektet, men begrundet af det var et krav lavede vi en opsætning klar til at vise konceptet for vores produkt.

Interfaces til UC1 i Java

Her satte vi interfaces op ud fra vores DCD for usecase 1 i Java. Her bruger vi klassens navn og tager de metoder der er med i, så de er klar til at anvendes til implementering senere.

Inception-milepæl

I inception-milepælen skulle vi have udarbejdet en overordnet vision for projektet, så vi vidste hvilket indhold vi skulle komme rundt om i vores opgave, have færdiggjort den så langt at den var klar til implementation i elaboration fasen og også skabt en prototype af en database, da dette er noget vi ikke har sat os ind i med den information der var tilgængelig og var derfor en udfordring der var overkommet. Vi valgte også at komme ud over milepælen med nogle af usecasene, men det var fordi vi havde udarbejdet til at have ekstra tid i vores projektplan, og besluttede os for at komme i gang med lidt mere end blot usecase 1.

Elaboration

2. Iteration(Steffen)

“Møde” med vejleder

Mødet med vejlederen var tid sat af til spørgsmål vi havde med hensyn til implementation af UC1, eftersom vi aldrig har arbejdet i tråde i Java, så vi havde behov for hjælp med at gøre det ordentligt.

Planlagt præsentation for holdet

Her havde vi en dag hvor alle teams skulle fremlægge hvad de havde lavet for milepælsplan, og havde derfor ikke noget tid til at arbejde, men fik fremvist hvor langt vi var nået og fik feedback af vejlederne.

Implementer UC1

Her implementerede vi UC1, hvor vi brugte de interfaces vi havde lagt ind fra 1. iteration, samt den ny fundne viden fra vores møde med en vejleder og fik en funktionel implementering

Implementer foreløbig database

Her implementerede vi en foreløbig version af vores database i selve programmet, som kunne anvendes til at få usecase 1 til at fungere som om programmet var sat ordentligt op.

Opdater DOM

Efter at have fået usecase 1 implementeret opdaterede vi vores DOM så den var tilpasset til programmet indtil videre.

Få visionsdokument godkendt

Her fik vi fremvist vores visionsdokument for vores vejleder og godkendt de forskellige dokumenter efter at have tilpasset nogle småting så visionsdokumentet bedre dækkede over vores projekt.

UC2 artefakter, interface, implementering

Eftersom elaboration-fasen går meget i dybden med analyse og design har vi en masse usecases hvor vi har udarbejdet alt det forebyggende arbejde og udarbejdet interfaces og implementation. Vi startede med UC2, hvor vi tilføjede dokumenter i samme rækkefølge som UC1 på trods af at vores projektplan siger vi startede med OC, for at sikre os at vi ikke sprang nødvendige skridt over, og sluttede af med at tilføje interfaces ud fra DCD og implementerede UC2 så den fungerede i vores program.

UC3 artefakter, interface, implementering

UC3 havde vi også på forhånd udarbejdet en formel usecase til, som vi løbende arbejdede på samtidig med UC2, så vi ikke kun havde gang i en enkelt ting af gangen, eftersom vi var to på vores team. Det vil sige at vi generelt arbejde side om side på de forskellige usecases, alt efter hvad der var behov for. Igen gennemgik vi de forskellige ting i rækkefølge på trods af vores projektplan viser OC i starten. Igen tilføjede vi

efterfølgende interface og anvendte metoderne i vores interface til implementation af en fungerende version af UC3. Efter vi havde implementeret UC2 og UC3 opdaterede vi vores DOM igen for at tilpasse den til hvad vi havde lavet indtil videre.

Beskriv UC 4-6 formelt

Eftersom vi nu havde implementeret UC1-3 var det tid til at beskrive flere usecases formelt, så vi kunne udarbejde artefakterne klar til implementering i næste fase, eftersom denne fase mere er analyse og design.

Opdater DOM

Igen tilpassede vi vores DOM efter at have tilføjet formelle usecases for UC 4-6.

UC4 artefakter

Med samme opbygning som tidligere usecases gik vi i gang med de forskellige artefakter for de næste 3 usecases. Vi startede med UC4, dog lavede vi også løbende på usecase 5 og 6 samtidig, hvor vi tilføjede aktivitetsdiagram, systemsekvensdiagram, operationskontrakter, opdaterede DCD til at være tilpasset og lavede et SD for usecasen.

UC5 artefakter

På samme måde som UC4 gik vi i gang med artefakter for usecase 5, som løbende blev lavet samtidig med usecase 4 og 6. Igen aktivitetsdiagram, systemsekvensdiagram, operationskontrakter, opdatering og tilpasning af vores DCD og et SD til usecase 5.

UC6 artefakter

Igen, ligesom UC4 og UC5 blev den opbygget på samme måde, løbende samtidig med de to andre usecases. Vi valgte dog ikke at lave SD for usecase 6, eftersom den mindede tilpasseligt meget om det SD der var lavet til UC5, så vi ikke brugte tid på noget der ikke ville være interessant dokumentation for vores program.

Planlæg 3. iteration

Til sidst i denne iteration efter at have gennemgået det meste af det analytiske og designmæssige bag vores program kunne vi planlægge til 3. iteration for projektet. Den 3. iteration er også en elaboration-fase, hvor

vi dog også går mere op i implementation, da meget af dokumentationen er gennemgået i den første del af elaboration-fasen. I den næste iteration blev der planlagt således for implementation af interfaces for de gennemgåede usecases, implementering sat op for de 3 nye usecases, samt tilretning af artefakter efter vi er kommet videre med det programmeringsmæssige i programmet. Udover det planlagde vi at lave artefakter og mockup for UC7, gennemgang af vores database og argumentation, gennemgang af aktuelle designmønstre samt lave iterationsplan for 4. og 5. iteration. Desuden havde vi planlagt at påbegynde rapport, så vi kunne lave den løbende som vi fik implementeret de sidste ting i de kommende iterationer. Alt dette er for at nå vores elaboration-milepæl.

3. Iteration (Martin)

Vores tredje iteration var vores anden iteration i vores elaborationfase. I denne tredje iteration gik vi efter UP's model og begyndte at implementere mere i forhold til før, samt at lave de interfaces der kunne laves i forhold til de artefakter vi havde konstrueret inden for OOA og OOD. Udover dette var der også visse milepæle der skulle overholdes udover vores generelle udvikling af systemet som der var sat ind i vores projektplan. Vi har bevidst valgt at ikke have tests med i denne iteration endnu, da vi ikke følte at programmet var testbart endnu. Vi havde et par metoder der godt kunne testes på, men vi ønskede at udvikle og implementere et par usecases endnu, inden der blev implementeret en decideret testsuite til programmet.

Interfaces

Vi begyndte at fremstille interfaces i vores udviklingsmiljø, for at klargøre til mere implementation af programmet. Denne iteration er, som man kan se i bilag 5, mere fokuseret på implementation end første iteration af elaboration, hvilket også afspejles af projektplanen, hvor man kan se at vi blot implementerede 1 stor, og 2 meget små usecases i elaborations iteration 1. Disse interfaces blev lavet direkte fra vores konstruerede DCD, hvilket forklarer den korte tid der var sat af til det.

Implementation af UC4

Efter dette gik vi i gang med implementationen af usecase 4, hvilket vi havde sat til at vare 1 dag efter erfaringen med usecase 1, der tog længere tid end beregnet. Alle vores implementations punkter i vores iterationsplanen var sat til at vare 1 dag grundet dette. Dette tog dog kortere tid end beregnet, og vi kom hurtigt videre til næste punkt på planen.

Tilret tidligere artefakter

Efter der var blevet arbejdet med artefakterne, og vi havde implementeret op til usecase 4, så valgte vi at sætte en pause på vores udvikling af programmet til at gennemgå alle vores tidligere artefakter for sporbarhed, samt for generelle syntaksfejl. Vi rettede også de ting til vi havde ændret i selve systemets logik da vi implementerede, da der nogle gange kan forekomme fejl i ens opsatte arkitektur som man laver inden man implementerer. Denne iterative udviklingsmodel, UP, gav os lov til dette på en nem måde[8] i forhold til hvis vi havde valgt vandfaldsmodellen, så dette var intet problem at indsætte som et punkt her.

Implementer UC5

Usecase 5 blev implementeret efter den tidligere tilretning, hvilket også viste sig at være en god idé, da vi brugte hele den tilsidesatte dag på at tilrette disse tidligere artefakter for designfejl, og fandt nogle fundamentale fejl, hvilket, hvis det var vandfaldsmodellen, ville have skabt nogle store oprydningsdage i kode og artefakter.

UC7 OOA/D

Da analyse og design stadig fylder meget i elaborations iteration 2, så valgte vi at lave hele usecase 7's OOA/D artefakter inden vi ville implementere usecase 6. Vi startede med den formelle usecase for denne, og efterfølgende lavede vi aktivitetsdiagrammet for denne usecase. SSD/OC kom efter AD'et, og dermed var OOA færdig for usecase 7. Vi lavede klassediagrammet til den baseret på OC'et og de andre artefakter, og valgte at hoppe over sekvensdiagrammet til denne usecase, da den hverken er stor eller interessant nok til at vi følte at der var behov for at have den med. Vores projekt havde begrænsede ressourcer, og da den blot var et databasekald ned på vores persistence pakke følte vi det ikke nødvendig, heller ikke når vi allerede havde et sekvensdiagram der viste næsten det samme i en af vores andre usecases.

Til sidst i udbygning af UC7 lavede vi et mockup af denne usecase i programmet Microsoft Paint for at derefter lave mockuppet i programmet SceneBuilder hvilket alle vores andre mockups også er lavet i. Det smarte ved dette er at der direkte kan kobles kode på denne mockup, hvilket gør det meget nemt at implementere mockuppet til implementationen. Dette punkt tog fra slutningen af onsdag d. 18 til midt på dagen om torsdagen, så udbyggelsen strakte sig over 2 dage, men tog i realiteten ikke 2 arbejdsdage at udarbejde, som der måske kan forstås på projekt-planen.

Implementer UC6

Usecase 6 blev implementeret efter usecase 5, og igen blev der sat en hel dag af til dette om torsdagen, og der blev arbejdet hjemme på opgaven indtil den var blevet færdig. Da den mindede meget om usecase 4

[8] [Philippe Krutchen] The Rational Unified Process An Introduction s. 55-57 & 76-77

blev meget fra denne genbrugt, og den tog i realiteten ikke lige så lang tid som der var planlagt til den, men efter implementationen af denne blev der finpudset på de andre implementerede usecases, og tilføjet eventuelle nice-to-haves såsom flotte fejlbeskeder til brugeren og lignende.

Gennemgang af database og argumentation

Dette punkt var et krav til milepæl b som var opsat af vore vejledere. Vores database skulle være i orden, og vi skulle have styr på argumentationen om hvorfor vores database var god i forhold til de standarder der normalt er opsat om databaser. Udover dette gennemgik vi vores datamodel for databasen, og færdiggjorde den så den var klar til at blive præsenteret for holdet. Dette punkt kan godt ses lidt som en form for review af vores database, da vi gik den igennem skridt for skridt, dog uden at dokumentere den som et review normalt bliver dokumenteret. Vi fik rettet databasen igennem, og kom også på et par problematikker som blev løst, såsom forkerte datatyper i databasen, samt en for dårligt udarbejdet databasemodel.

Gennemgang af aktuelle designmønstre

Da milepæl b også indeholdte det punkt at vi skulle snakke om de designmønstre vi havde valgt at bruge i projektet, så samlede vi alle de designmønstre vi havde diskuteret os frem til at have brugt i vores program, og fandt også eksempler på disse heri. Af designmønstre kan der nævnes at trelags modellen blandt andet blev brugt, men der vil være mere om dette i produktbeskrivelsen, hvor disse designmønstre bliver gennemgået.

Iterationsplan for iteration 4

Da vi havde planlagt at lave en iterationsplan senest i slutningen af hver foregående iteration blev iterationsplan for iteration 4 planlagt og lavet i denne iteration på dette tidspunkt. Denne iterationsplan var også krav til milepæl B.

Iterationsplan for iteration 5

Vi valgte at lave den sidste iterationsplan i denne iteration også, da vi havde et godt billede af hvordan vores transition skulle foregå. Den er en meget lille og simpel iteration, så derfor planlagde vi den med de ting som vi skulle lave til sidst, lige inden projektet bliver overdraget til vejlederne.

Påbegynd rapport

Dette punkt var med til at sikre at vi påbegyndte rapporten på det daværende tidspunkt, da vi havde sat som mål at skrive et vis antal sider hver dag, for at sikre os at rapporten blev skrevet i god tid, så den kunne opnå en vis form for kvaltiet, i forhold til hvis den blev skrevet i sidste øjeblik.

Elaboration milepæl

Målet med en elaboration fase et blandt andet at der skal forefindes en eksekverbar prototype af vores program, og dette opnåede vi med hjælp af de aktiviteter vi havde planlagt i vores iterationsplan. Udover dette fik vi også nået milepæl B som var opsat af vores vejledere, samt en gennemgang af tidligere artefakter for at kvalitetssikre disse internt imellem os på udviklingsholdet.

Construction

4. Iteration

Planlæg review for OOD artefakter

Da vi gerne ville kvalitetssikre vores OOD-artefakter på en god og professionel måde, planlagde vi som det første i denne iteration et review for de OOD-artefakter vi havde produceret igennem projektet. Vi planlagde hvilke roller der skulle være med til dette review, og hvilket et tidspunkt det skulle foregå. Vi gik også artefakter igennem hurtigt for at skimme igennem hvilke vi gerne ville fokusere på.

Implementer UC7

Implementationen af UC7 foregik efter hensigten da vi sad og implementerede den, og da constructionfasen jo er navngivet efter at man konstruerer programmet i denne fase er det oplagt at have de sidste usecases med der skulle implementeres. Dette er også derfor vi sad og implementerede denne så tidligt i fasen. Igen var der sat én dag af til dette, selvom det reelt varede nogle timer.

”MARTIN EFTER SKOLE/STEFFEN EFTER SKOLE”

Da vi begyndte at kunne mærke at projektet var blevet stort og at deadlinen var om et kort stykke tid begyndte vi at planlægge hjemmearbejdet ind i projektplanen for at få sikret os at det blev lavet på det pågældende tidspunkt. Vi følte ikke at projektet var ved at slippe væk, men vi kunne godt lide at vi hurtigt kunne smide et par arbejdsopgaver ind i projektet som vi individuelt skulle sidde derhjemme og lave. Autodatabasen var et krav i opgaveformuleringen at den skulle sætte sig automatisk op, så derfor blev dette arbejdet på derhjemme i stedet for inde på skolen. Brugertest planlægning blev også lavet

derhjemme, da vi gerne ville have at det stod parat hvis vi stødte på nogle brugere der gerne ville teste programmet imens vi sad inde på skolen.

Påbegynd Java Doc af programmet

For at vise at vi har styr på Java Doc valgte vi at lave Java Doc for de vigtigste metoder i vores applikation. Vi påbegyndte dette inde på skolen, og selve opstarten af Java Doc var sat til at vare 2 timer hvilket vi overholdte. Vi valgte at ikke bruge for lang tid på den nu, da der kunne forekomme ændringer i programmet, og denne Java Doc skulle dermed ændres hvis dette skete.

Brugertest af foreløbigt program

Da vi umiddelbart ikke kunne finde nogle at teste programmet på inde på skolen valgte vi at teste programmet ude for skolen ved folk vi allerede kendte. Den blev testet på forskellige aldersgrupper, og vi fandt også ud af eventuelle fejl vi skulle rette på i programmet. Denne brugertest er også en del af UP's model at disse tests skal forekomme på nuværende tidspunkt i faseplanen, dermed var det oplagt at påbegynde vores tests af programmet.

Formel UC8-10

Da der stadig, ifølge UP-modellen, forefindes analyse og design af programmet i starten af construction fasen valgte vi også at færdiggøre analyse og design delene af programmets sidste par usecases. Disse usecases var ikke så relevante for færdiggørelsen af projektet, men stadig meget relevante for kravspecifikationerne af programmet. Vi valgte at først færdiggøre vores formelle usecases på samme tid lige efter hinanden, som vi igennem projektet har gjort samlet, så vi senere kunne sidde og arbejde for os selv med de sidste artefakter for at være mest effektive.

UC8 OOA/D

Som der kan ses på vores projektplan er der kun det punkt med til UC8 der hedder implementation. Dette er grundet denne usecases store lighed med UC1, hvor der skal bestilles en kørsel for brugeren der er logget ind. Den eneste forskel på disse to usecases er at UC8 modtager et brugernummer fra et tekstfelt manuelt fra MidtTrafik-brugeren, i forhold til usecase 1 hvor den automatisk tages fra hvilket et

kundenummer brugeren loggede ind med. Vi tog dette bevidste valg da vi lavede den formelle usecase til den, og rettede projektplanen derefter da vi ikke vidste det før vi sad og arbejdede med den.

Rettelse af brugertest feedback i programmet

Da vi nu havde fået feedback af brugertestene vi havde kørt, valgte vi at rette programmet til på dette tidspunkt for at få det ordnet inden vi gik videre med implementationen af nye usecases. Dette tog kun cirka en time som beskrevet i projektplanen, da det var mindre småfejl der blev rettet rent kodemæssigt.

UC9 OOA/D

UC9s artefakter blev lavet fuldt ud, for derefter at blive implementeret. Det var en meget simpel usecase, den krævede dog konstruktionen af en uforudset brugergrænseflade i form af kommentarfeltet der kommer frem når man højre-klikker på en kørsel. Denne var dog en simpel brugergrænseflade at designe og implementere, så der var ikke noget skrid i tidsplanen.

UC10 OOA/D

Vi valgte fra at lave artefakter til UC10 da den var en så simpel usecase at beskrive da vi ikke følte den ville have noget værdi for hverken os eller læserne af dokumentationen. Usecasen består af at der bliver tilføjet en lille knap til et af skærmbillederne, hvorfra man så kan trykke på den og sortere alle kørslerne til at kun vise de afholdte kørsler. Vi havde allerede vist sortering og databasekald i andre artefakter vi har konstrueret i dette projekt. Havde vi derimod haft lidt flere ressourcer i form af flere på teamet ville vi højest sandsynligt have sat nogen på til at lave disse artefakter, blot for en sikkerheds skyld.

Implementation af UC10

Da UC10 var sådan en lille usecase at implementere tog denne selvfølgelig heller ikke en hel dag at implementere som der stod på projektplanen at der var sat af til den. Tiden der var tilbage blev brugt til at finpudse programmet i form af småting vi synes der kunne være bedre internt, såsom bedre fejlbeskeder der var vist til brugeren osv.

Review af OOD-artefakter

Vi satte os for at reviewe OOD-artefakterne internt i vores gruppe, uden andre roller, da vi desværre ikke kunne få fat på andre der kunne være med grundet tidsmangel. Vi brugte 2 timer og kom så langt at vi fuldt ud reviewede alle artefakter til usecase 1, hvilket vi var tilfredse med da den var en meget vigtig usecase at skulle have styr på. De andre artefakter aftalte vi at vi ville kigge igennem for sporbarhed når vi havde

huller i tidsplanen eller hvis vi blev færdige med ting før tid, hvilket vi også gjorde på et senere tidspunkt, hvor vi så gik dem alle igennem og fik dem rettet godt igennem for sporbarhedsfejl.

jUnit-tests

Da construction fasen byder på en stor del tests af programmet gik vi selvfølgelig i gang med de jUnit tests vi mente vi ville have med ud fra programmet, efter at den sidste usecase var implementeret, og at de vigtigste artefakter var kigget igennem. Vi valgte at have jUnit tests på de vigtigste metoder og klasser, for ikke at overbebyrde os selv med jUnit tests af hele programmet, selvom det jo selvfølgelig ville have været mest optimalt. Igen, hvis vi havde flere ressourcer i form af flere på teamet var dette noget vi ville have arbejdet videre med.

Projektskrivning

Projektskrivning blev endnu engang sat på skemaet, hvilket vi lavede i weekenden, imens vi sad og kommunikerede online. Dette hjalp os til at få lavet projektskrivningen i god tid, så der ikke blev travlt i sidste øjeblik.

Afsluttende brugertest/finpudsning

Der blev sidst i construction fasen lavet den sidste brugertest for at fange de sidste par fejl der end måtte være som brugere kunne synes værende trælse, og bagefter dette blev disse finpudset. Der var ikke så mange fejl tilbage som de brugere vi testede det på opdagede, så derfor tog disse to ikke så lang tid at færdiggøre.

Milepæl for construction fasen

Milepælen for construction fasen bød på et fuldstændigt kørende program der var testet igennem, samt medfølgende artefakter udført for dette program i form af systemudviklingsartefakter vi lavede inden programmet blev implementeret. Denne fase bød på en stor mængde implementationer og tests, hvilket gjorde at vores program til MidtTrafik blev færdiggjort med de udspecificerede kravspecifikationer vi havde sat op i starten.

Transition(Martin)

5. Iteration

Transition

Vores transition fase er meget kort, da vi ikke havde den store tid til en stor beta test hos modtageren, da modtageren jo først modtager vores endelige program når hele projektet er afsluttet i form af en afleveret

opgave. Vi har dog stadig nogle vitale opgaver der skal laves i denne fase, og som der beskrives i de efterfølgende punkter.

Eksportering af .jar til aflevering

For at kunne udgive vores produkt videre skal det eksporteres til en .jar fil der kan køres hos slutbrugeren. Udover dette skal denne .jar fil indeholde alle de libraries vi har brugt i projektet, hvilket vi også tilføjede til den endelige .jar fil. Dette punkt tog ikke meget længere end en time, nærmere en halv time, og vi stødte ikke på yderligere problemer ved dette.

Dokumentation for opsætning af DB til opstart

I vores projekt har vi valgt at have en automatisk database til de vitale ting i programmet, nemlig til den normale struktur. Vi har dog fravalgt at have vores regionsdatabase med, da den fylder 1400+ dataindsætninger, og vi følte at det ville være for meget at indsætte i vores kode til at det ville give mening. Derfor er der dette punkt, der forklarer hvordan man på 30-60 sekunder kan sætte databasen fuldt op på sin egen pc. Det er klart, at hvis det var en normal slutbruger dette program skulle ud til, at vi ville have inkluderet dette i setuppet, men opgaveformuleringen gav os dette valg til at inkludere databasen med en manual til opsætning, så derfor greb vi denne chance for at kunne undgå for meget spildt plads i koden til ting som disse.

Afsluttende Java Doc

For at gøre vores programt nemt for andre at kunne videreudvikle på valgte vi at lave mere Java Doc af de vitale metoder i programmet. Dette punkt tog lidt mere end 2 timer, men vi kom let omkring det, da det blot var en forklaring af de vitale metoder/klasser inde i selve koden. Vi valgte at kun skrive Java Doc til vores interfaces, da det normalt er best practice at gøre dette, med mindre at ens implementation viger ekstremt meget fra interfacet/API 'en.

Fremstilling af FAQ samt Wiki til .git

For at gøre vores projekt endnu mere brugervenligt for eventuelle udviklere der kunne tænke sig at videreudvikle på vores produkt, eller at bruge det til inspiration valgte vi også at lave vores GIT-side mere

brugervenligt i den form af at en README side blev tilføjet med de væsentligste ting der var værd at vide om vores projekt.

Dataordbog

Vi havde lavet den fejl at ikke lave vores dataordbog tidligt i projektet da vi på teamet var så få til at udvikle at vi tit var på "samme side" når vi snakkede om projektet. Denne ulempe gjorde at vi manglede et relativt vigtigt artefakt i vores projekt, som vi valgte at lave færdig inden afleveringen af projektet. Vi havde ikke lidt af manglen på dette artefakt, men nytilkomne udviklere kunne godt lide under det, da de ville være i tvivl om de diverse begreber som der forekommer under koden/i artefakterne der var udarbejdet.

Afslut projekt

Vores afslutning af projektet er sat til at vare hele afleveringsdagen, da vi gerne ville være i god tid med afleveringen, samt for at dække for eventuelle deadlines der ikke blev overholdt, eller sidste øjebliks vitale ændringer/tilføjelser som vi end måtte have i projektet. Vores risikoanalyse nævnte jo også et par ting der kunne have indflydelse på afleveringen, så derfor ville vi være på den sikre side med at vi fik afleveret til tiden, ved at sætte hele denne dag af til det.

Milepæl for transition

Milepælen for en transition er en udgivelse af vores produkt[9], hvilket vi jo har gjort i den form af at det er afleveret med alle dens artefakter til vores vejledere, klar til at blive bedømt. Denne milepæl nåede vi uden større problemer, da denne rapport jo er en del af det samlede færdige produkt.

Produktbeskrivelse

Indledning til UC1 (Steffen)

Grunden til vi har valgt at gå mest i dybden med usecase 1 i forhold til produktbeskrivelsen er at den indeholder den største og vigtigste del af hele vores endelige program, altså Bestil Kørsel. Den indeholder tråde under udregning af pris, den bruges til at oplyse alle informationer for kørsel til programmet når de videresendes, og er den usecase som programmet mere eller mindre omhandler ved brug. Den indeholder både prisen, adressen, tidspunktet, dato, antal personer, antal hjælpemidler, antal bagage og en kommentar til MidtTrafik. Det er alle vigtige informationer for vores program, da vores andre usecases

[9] [Philippe Krutchen] The Rational Unified Process An Introduction s. 75

anvender disse informationer. Det er den vigtigste grund til at vælge denne usecase, udover det faktum at den har masser af indhold at forklare om.

Formel usecase

Først og fremmest er succesgarantien i vores formelle usecase at brugeren får registreret sin kørsel og den bliver sat klar til godkendelse i systemet. Desuden skal brugeren have besked om den succesfulde bestilling.

For hovedscenariet er punkt 1 bare interessepunkt, hvor brugeren gerne vil anvende bestillingssystemet. Punkt 2 er så hvor brugeren anvender programmet til at indtaste de nødvendige oplysninger for at udregne en pris til sin kørsel med flextrafik. Punkt 3 anmoder brugeren om at få udregnet en pris for sin kørsel, og i punkt 4 reagerer systemet på anmodningen og validerer oplysningerne, og hvis oplysningerne indeholder fejl i forhold til udregning af prisen giver punkt 4a en fejlmeddelelse og går tilbage til punkt 2 for at genindtaste oplysninger. Hvis oplysningerne ikke er forkerte udregner systemet en pris for brugerens kørsel i punkt 5. Da dette kører som en tråd så brugeren kan arbejde videre på sin bestilling samtidig med at programmet udregner, giver programmet besked om at den er i gang med at udregne prisen for brugeren. Når systemet har færdigudregnet prisen for kørslen viser den prisen for brugeren i systemet ved punkt 7. I punkt 8 indtaster brugeren så de manglende oplysninger såfremt han ikke har nået det under udregningen af pris, og i punkt 9 accepterer han prisen ved at anmode om at få registreret sin bestilling til systemet. Systemet validerer så igen informationerne, dog for hele kørslen denne gang, og sikrer sig at brugerens informationer er korrekte. Hvis der mangler informationer eller de er ugyldige giver systemet en fejlbesked i punkt 10a og specificerer hvad den ugyldige/manglende information er, og fortsætter så hovedscenariet fra punkt 8 igen. Hvis brugeren har indtastet informationerne korrekt registrerer systemet den bestilte kørsel i sit system i punkt 11, og i punkt 12 giver systemet en besked til brugeren om at registreringen af kørslen er blevet gennemført.

Brugergrænsefladen

Vores brugergrænseflade for usecasen Bestil Kørsel er sat op efter at være så overskuelig som mulig på trods af at rigtigt meget information skal udfyldes. Vi startede med et mockup hvor vi nogenlunde gik igennem idéer til hvordan opsætningen kunne være i selve programmet, og kom frem til at det ville være en god idé at vælge at sætte startadressen og alle informationer på en enkelt linje, og så slutdestinationen sat op på samme måde nedenunder, så man kategoriserer dem sammen. Da alt der ligger over prisen har relevans for selve kørslen med tid, dato og destination valgte vi også at sætte det op så tiden på startdestination bliver specificeret lige under startdestinationens linje, og dato og antal kilometer der bliver kørt bliver smidt ind nedenunder slutdestinationen. Under dette valgte vi at sætte prisen, for at dele siden i to så der er nemt at overskue hvilke informationer der var nødvendige at udfylde for at kunne udregne

prisen. Det der ligger under prisfeltet er så resterende information om selve bilen, altså hvor mange personer der er med på turen og hvor mange pladser der skal fyldes op med bagage og hjælpeudstyr. Aller nederst giver vi en sidste mulighed for at man kan skrive en kommentar til admin så de kan se hvis der er nogle specielle behov brugeren har for kørslen der skal tages hensyn til. Under kommentarfeltet dukker det endelige bestillingsfelt så frem når man har fået udregnet prisen, så man ikke er i tvivl om at man ikke kan bestille uden først at have fået en pris fastslået.

Vi har desuden valgt at lave videre til det aktuelle program fra mockup med JavaFX, eftersom FXML filer giver en god opdeling mellem selve designet fra vores GUI med dens controller, så det ikke ligger i samme klasse. Desuden gør det også at det er nemt at sætte metoder ind for de forskellige knapper og felter, hvor vi har valgt at bruge ”haandter” til de forskellige nødvendige felter. Dvs. hvis vi f.eks. skulle anvende en knap for at udregne prisen kalder vi den bare haandterUdregn, og kobler det på inde i JavaFX programmet så det også står i koden. Vores GUI controller håndterer derved det der sker ude i brugergrænsefladen og skaber en kobling imellem brugergrænsefladen og vores logik ifølge controller mønstret i GRASP.

Domænemodel

Vores domænemodel kigger på de koncepter og begreber der er i spil i hele vores system. Det viser hvordan de forskellige koncepter og begrebers sammenspil fungerer, og de forskellige funktioner der er imellem dem, altså alt hvad de hver især indeholder eller sender videre til andre koncepter. Vores domænemodel er sat op til at minde meget om et klassediagram, dog er det her mere en idé frem for det tekniske af programmet. Det er lavet alene for at skabe overblik over hele programmet, og hvordan de yderligere systemudviklingsartefakter skal laves, da domænemodellen viser sammenhængen mellem de forskellige koncepter. Ud fra domænemodellen udarbejdede vi vores prototype til databasen hvilket kun har haft meget få ændringer siden første forsøgte implementation. Farveopsætningen af vores domænemodel er lavet for nemmere at kunne se hvordan sammenspillet foregår imellem de forskellige koncepter og begreber, så de forskellige pile er sat op til at passe med at hovedkoncepterne sender dem videre eller får dem fra andre. Det vil sige alle pile der er sat til eller fra brugeren er farvekodet som rødt, alt der kommer til eller fra MidtTrafik er blåt, alt andet der kommer til og fra kørsel er gult, og resten er bare sat op som en standardfarve. Domænemodellen kan findes bilag 6.

Aktivitetsdiagram

Vores aktivitetsdiagram viser mere eller mindre hvordan vores formelle usecase er udarbejdet, da pointen i AD er at vise en diagramform for vores usecase, dog er dataformer imellem tilføjet så man har en bedre idé om hvad der foregår imellem anmodninger frem og tilbage mellem system og bruger. Altså, når brugeren

har udfyldt prisoplysninger bliver der sendt Prisoplysninger videre som datatype til systemet, og når systemet oplyser pris for kørsel bliver datatypen Pris sendt tilbage til brugeren.

SSD

I vores systemsekvensdiagram bygger vi videre ud fra vores AD og formelle usecase. Her sender brugeren først et kald ”getPrisTilbud” over til FTP som FTP så validerer for at finde ud af om der er ugyldige oplysninger. Hvis den finder ugyldige oplysninger stopper programmet, ellers fortsætter den ned i diagrammet. Hvis den fortsætter i programmet sender FTP kaldet ”beregnPris” til flextur_sats. Derefter bliver en besked til brugeren sendt om at FTP udregner pris med ”beregnerPrisBesked” imens at flextur_sats arbejder. Når flextur_sats har udregnet prisen, sender den et returkald til FTP ”kørselsPris” som FTP sender videre til brugeren med samme returkald bagefter. Når brugeren har fået sin pris sender han et kald ”accepterPris” med sine informationer som parametre til FTP, som igen validerer oplysningerne. Hvis oplysningerne er ugyldige sender den ”forkerteOplysningerException” som returkald til brugeren og stopper programmet, ellers kører programmet videre. Dernæst kalder FTP ”gemKørsel” på sig selv, og sender et returkald ”kørselsBekræftelse” tilbage til brugeren.

DCD

Vi har valgt at dele vores DCD op i to diagrammer, et for at vise forholdene imellem klasserne, og et andet for at vise indholdet af de klasser der indgår i DCD’et. Disse to kan findes i bilag 7 og bilag 8. I selve indholdet af klasserne har vi valgt kun at vise hvad der er i vores interfaces, da vi vil have åbne muligheder for at redigere i datakernerne uden at vi løbende skulle opdatere vores DCD hele tiden. Klassediagrammet giver dog stadig et godt overblik over vores program, specielt da den er delt op i de to forskellige diagrammer. Klassediagrammet viser hvordan vi bl.a. har anvendt trelagsmodellen, da vi har et logiklag der kalder ned på vores database og de er separeret fra hinanden, vores præsentationslag er dog ikke vist i klassediagrammerne da vi stadig vil have åbne muligheder for at redigere i præsentationslaget uden løbende at skulle konstant opdatere. Vores præsentation har heller ikke interfaces, så det ville ikke passe ind at sætte dem i vores DCD da der kun vises interfaces. Vores farvekodning i diagrammet med forholdet mellem klasserne er sat op så man kan se alle pile der kører i og fra logiklaget, da de alle er lavet røde, og alle interne pile i persistence-laget er sat til at være orange. Alle pile der kører fra persistence-laget til domænelaget er farvekodet til at være lyseblåt. På den måde kan man nemt overskue hvilke piler der hører til hvilke domæner originalt, og kan nemt se når pilene i få tilfælde går over hinanden. Diagrammet er også en god måde at vise hvordan alt starter fra vores FTPController som sætter gang i hele programmet. Denne controller er også vores primære kobling til præsentationslaget, hvilket igen er et godt eksempel på trelagsmodellen. Vores diagram opsat efter interfaces viser bare alle de metoder der skulle implementeres,

så man let kan genimplementere vores kode i form af et API. Grunden til at vores kørselshistorik-klasse er så stor er ikke fordi vi ikke har anvendt high-cohesion fra GRASP mønstret, men fordi at vores TableView i vores præsentation bruger en domæneklasse til at bygge dens felter op, så resultatet er at vi har brugt den store domæneklasse til at bygge vores TableView op, som gør den bliver lidt uoverskuelig. Et eksempel på hvordan vi har anvendt high-cohesion er hvordan vi bl.a. har delt vores StartDestination og SlutDestination fra Koersel-klassen ud i deres egen domæneklasse for at skabe mere overskuelighed.

SD

getPrisTilbud

Dette SD er lavet direkte ud fra OC1 og andre informationer fra de tidligere artefakter. Først kommer der et metodekald fra getPrisTilbud på FTPController. Dette gør at FTPController skaber Validator, som laver en alt frame hvor den tjekker om validerInformationer er lig med true. Hvis det er true fortsætter programmet, ellers kommer en InvalidInformationException som et lost kald og slutter sekvensdiagrammet. Hvis programmet har fortsat forbi validationen skaber FTPController PrisBeregner, og så kalder FTPController en run metode på PrisBeregner. Her går de begge i tilstand for beregning, hvor PrisBeregner kører beregnPris på sig selv og notifyObservers på sig selv. FTPController kalder så getPris på PrisBeregner.

accepterPris

accepterPris er også lavet ud fra OC1 og de tidligere usecase 1 artefakter. Den laver et metodekald med accepterPris på FTPController, som starter en try/catch frame, hvor den prøver at kaste validerInformation på Validator, og catch fanger InvalidInformationException og sender som lost kald og slutter. Hvis der ingen exception blivet fanget i catch kører FTPController gemKørsel på KoerselsKartotek.

Tråde i vores program (Martin)

```
@Override
public void update(Observable o, Object arg) {

    if(pb.getTilstand() == Tilstande.BEREGNINGSFEJL){
        this.pris = -1.0;
    } else {

    this.pris = pb.getPris();
    tilstand = Tilstande.BEREGNET;
    }
}

@Override
public void getPrisTilbud(StartDestination startdestination, SlutDestination slutdestination, double km, Date dato)
        throws SQLException, UnknownKommuneException, InvalidInformationException {
    Validator validator =  new ValidatorImpl();
    validator.validerTilbud(startdestination, slutdestination, km, dato);
    pb = new PrisBeregnerImpl(startdestination, slutdestination, km,dato);
    pb.addObserver(this);
    try{
    (new Thread(pb)).start();
    } catch(Exception e){

    }
}
```

Figur 4 Kaldet på vores prisberegner som en ny tråd

For at kunne gøre brugeroplevelsen så problemfrit som muligt og uden afbrydelser/frysninger i programmet valgte vi at bruge tråde i vores program for at imødekomme dette problem. Som man kan se på figur 4 starter vi først med at validere informationerne i en instans af vores validator klasse, for at derefter erklære vores pb (som er sat til at være class scope for at den kan bruges i vores update også) med de informationer der netop er valideret. Denne valideringsmetode kan kaste en invalidInformationException, hvilket også ses på hvilke exceptions den kaster. Hvis denne bliver kastet så når den ikke ned på at instansiere en ny PrisBeregner med de informationer den skal have med i sin constructor. Efter dette bliver controller klassen sat til at være en observer på PrisBeregner, og der prøves på at lave en ny tråd med PrisBeregner der bliver startet med sin startmetode. Når start bliver kaldt bliver run også kaldt i den pågældende tråd, og i denne run metode bliver der kaldt en udregning af satsen.

Når denne sats er blevet beregnet, så notifier den sine observers, hvilket i dette tilfælde er controlleren, der så tjekker om der er sket en beregningsfejl (det tjekker den på den tilstand som prisberegnerklassen er sat til at have) og har den det, så sætter den prisen til at være -1, hvilket præsentationen kan håndtere. Vi har valgt at lade præsentationen håndtere hvad der skal vises, da det jo kunne være at man ville pålægge en ny brugergrænseflade på logikken der håndterer en prisfejl på en anden måde. Grunden til at vi sætter en prisberegningsfejl til at være pris=-1, vil blive forklaret i næste afsnit. Hvis tilstanden derimod ikke er en beregningsfejl fra prisberegneren, så sætter den prisen til at være prisen udregnet fra prisberegneren.

Håndtering af denne tråd i vores brugergrænseflade

For at kunne opdatere en brugergrænseflade i JavaFX kan man ikke anvende andre tråde end en specifik JavaFX tråd. Vi forsøgte på mange måder at få dette til at lade sig gøre, men det kunne ikke fungere på nogen måde. Efter lang tids søgen på Stackoverflow.com, samt en snak med vores vejleder fandt vi frem til en metode for at kunne undgå denne afgrænsning i JavaFX.

```
new Thread(new Runnable() {
    @Override
    public void run() {
        ftp.setPris(0);
        while (ftp.getPris() == 0 && kmFelt.getText()!="") {
            try {
                Thread.currentThread().sleep(1000);
            } catch (InterruptedException e) {
                prisUdregnet = false;
                ikkeAendreFelt();
                advarsler.ukendtFejl().showAndWait();

            }
        }
        Platform.runLater(new Runnable() {

            @Override
            public void run() {
                if(ftp.getPris() == -1.0){
                    progressBar.setVisible(false);
                    prisLabel.setText("0 Kr");
                    advarsler.postnrUkendtFejl().showAndWait();
                } else{

                prisLabel.setVisible(true);
                progressBar.setVisible(false);
                prisLabel.setText(String.valueOf(ftp.getPris()));
                accepterKnap.setVisible(true);
                }
            }
        });
```

Figur 5 while løkken og Platform.runLater metoden til udregning af pris

Vi skulle lave en while-løkke der hele tiden spurgte controlleren om en pris var udregnet, dette kan ses på figur 5. Det første vi dog gør er at lave en ny tråd inde i vores JavaFX klasse, hvilket gør at vi kan modificere JavaFX objekter herfra. Vi sætter prisen i ftp controlleren til at være 0 fra starten af, da vi senere spørger ftp controlleren om den er 0 i while løkken. Hvis den er 0, så venter vi 1 sekund for at derefter spørge igen. Hvis den derimod ikke er 0, så kalder vi den specielle metode der hedder Platform.runLater, der gør at noget kode bliver "sat i kø" til at blive kaldt når der er tid. Dette sikrer at vi ikke er i konflikt med en anden tråd der ikke er en JavaFX tråd, og fra denne Platform.runLater kan vi dermed sætte pris-labelen til den udregnede pris. Det er også heri hvor vi spørger efter om prisen er -1, hvorfra vi så genstarter prisen i pris-labelen, og siger til brugeren at der er sket en fejl. Det er kun fordi at det er JavaFX objekter vi modificerer at denne løsning skulle anvendes. Var det ikke for det, kunne vi nemt have nøjes med en meget simplere løsning.

Brug af ekstern .jar fil flextur_sats

Et af kravene fra opgaveformuleringen var at den eksterne .jar skulle bruges til at udregne prisen fra. Som man kan se på figur 6 vises der metoden beregnPris fra PrisBeregner klassen, der bliver kaldt i dens run metode. I denne starter den først med at finde startkommunen der er indtastet baseret på det indtastede postnummer. Vi har udviklet noget kode til at slå kommunen op i vores regionskartotek

```
@Override
public double beregnPris(StartDestination startDestination, SlutDestination slutDestination, Date dato)
        throws UnknownKommuneException, SQLException{
    KommuneKartotekImpl kk = new KommuneKartotekImpl();

    String startKommune = kk.postnummerTilKommune(startDestination.getPostnummer());


    startKommune = startKommune.replace(" Kommune", "");
    startKommune = startKommune.replace("-", "/");
    String slutKommune = kk.postnummerTilKommune(slutDestination.getPostnummer());
    slutKommune = slutKommune.replace(" Kommune", "");
    slutKommune = slutKommune.replace("-", "/");

    Sats s = Sats.i();
    tilstand = Tilstande.BEREGNET;

    sats = s.getSats(startKommune, slutKommune, dato.getYear(),dato.getMonth(),dato.getDay());
    setChanged();
    notifyObservers();

    return sats;
}
```

Figur 6

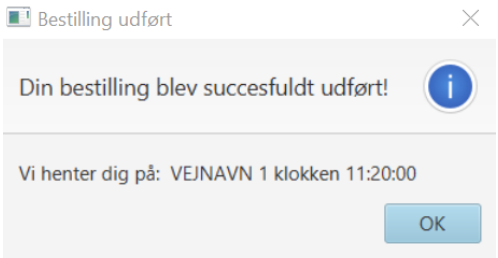
hvilket vi har opbygget ved hjælp fra en csv fil fra Post Danmarks hjemmeside der har alle Danmarks kommuner sammenkoblet med alle postnumre i Danmark. Efter dette formaterer vi kommune navnet vi har slået op til at passe med .jar filens formatering af kommuner, og gør det samme for slutkommunen. Vi laver derefter en ny instans af Sats klassen der ligger i flextur_sats filen, og denne bliver instansieret som en Singleton, da den er defineret til at være dette i klassen vi fik udleveret. Det har den fordel at der kun kan instansieres 1 instans af objektet af gangen. Vi sætter herefter tilstanden som værende beregnet, da hvis der ville forekomme en fejl ville tilstanden blive sat som værende en beregningsfejl alligevel. Efter dette sætter vi sats til at være den værdi vi udregner fra Sats klassen, med start og slutkommune, samt den pågældende dato for kørslen. Når denne er blevet udregnet notifier den dens observers, hvilket betyder at ftp-controlleren kan få at vide at den skal hente prisen, for derefter at brugergrænsefladen får dens pris at vide.

Den eksterne .jar fil er pakket ind i vores endelige .jar fil der blev afleveret som et eksternt library som vores program bruger.

Håndtering af exceptions i vores program (Martin)

Generelt set har vi håndteret alle de exceptions som programmet kunne have i sinde at kaste i tilfælde af en fejl i koden. Vi har et par steder valgt at lade dem gå ubemærket hen for brugeren i den form af f.eks. en NullPointerException i vores export metode, hvor at hvis brugeren ikke har valgt at gemme filen ved at trykke på det røde kryds i hjørnet af programmet så smider den en NullPointerException hvilker vi fanger, men vi gør intet med den. Brugeren skal jo have lov til at fortryde sin eksport til csv.

Fejlbeskederne i vores program er meget lette at forstå for brugeren, og vi har valgt at bruge den indbyggede pop-up beskeds funktion i JavaFX der hedder ”Alert”. Med denne har vi lavet en klasse i præsentation hvori de forskellige muligheder for advarsels beskeder er, som vi kalder ” Advarsler” klassen. I denne definerer vi overskrift, titel samt en brødtekst til beskeden som brugeren får at se, og derefter kalder vi den i den pågældende præsentationsklasse det er nødvendigt i. Dette har gjort vores kode en del mere overskueligt i forhold til at hvis vi definerede dem inde i de pågældende klasser, i stedet for at definere alle i Advarsler klassen. Vi har også ved siden af lavet en Information klasse med de beskeder der kommer frem når man f.eks. har bestilt en kørsel.



Figur 7 Eksempel på den indbyggede Alert funktion i JavaFX, denne med AlertType "Information"

Databasen i vores applikation (Martin)

```
GET_ENKEL_TID_AFHOLDT = "SELECT * FROM koersler WHERE kundenummer = ? AND tidspunkt>= ? AND tidspunkt <= ? AND tidspunkt < NOW() AND GODKENDT_KØRSEL = true";
```

Figur 8, en SQL streng. Kan ses i vores koerselskartotek klasse.

Databasen i vores applikation er udarbejdet i det lille program der hedder HSQLDB. HSQLDB kører primært på en SQL syntaks med mindre variationer, og er et letvægtigt værktøj der let kan integreres med Eclipse. Vores primære databaseopbygning kan forefindes i vores databasemodel i bilag 9, samt en dataordbog der forklarer de forskellige datatyper der kan forefindes i bilag 10. Stort set er vores database bygget op fra domænemodellen, hvor vi har brugt de begreber der kunne findes derfra, samt de relationer vi fik fra denne domænemodel. Vi lavede vores første databaseprototype allerede i vores inception fase, og vi har bygget videre på denne første prototype til nu hvor den endelige er færdig. Vi har, angående databasetilkobling i vores kode, brugt vores persistence pakke til at placere alle database relevante klasser i. Vi har også brugt PreparedStatements i vores tilkoblinger, for at øge sikkerheden i vores database tilkobling, så der ikke kan bruges metoden der hedder SQL injection til at redigere vores database fra den ydre tilkobling fra brugergrænsefladen. Vi har også valgt at bruge en tidligere database tilkoblingsklasse som vi har brugt i andre projekter, hvilket er den der hedder DataAccess, som der blot returnerer en connection som der kan bruges til de andre database klasser. Denne connection bliver instansieret hver gang der er en ny databasemetode med andre ting, for at netop undgå at vi genbruger den samme connection flere gange i samme klasse, som der blot ville skabe problemer. En måde vi kunne have omgået dette på ville dog også være ved at have lavet en Singleton klasse på denne, men da vi sørgede for at vi ikke genbrugte den samme connection flere gange, samt at vi lukkede for denne connection i slutningen af hver metode følte vi at vi var godt dækket ind. En af de mest interessante databasemetoder vi gerne vil fremhæve er metoden der bruger SQL strengen på figur 8. Dette SQL kald kalder de kørsler vi skal have for en angivet bruger, inden for et angivet tidsinterval, og for at gøre koden mindre kompleks i logiklaget vælger vi at kun hente det data ned som vi beder om direkte fra databasen. Vi føler at denne metode er interessant i forhold til de andre database kald, da den bruger selve SQL 'et til at hente og sortere sine data inden de bliver hentet ind i programmet.

Java Doc udarbejdning (Martin)

For at dokumentere vores program på en kvalitativ måde har vi valgt at bruge den indbyggede Java Doc funktion der er integreret i Eclipse udviklingsværktøjet. Med Java Doc kan vi nemt komme omkring at kommentere de mest relevante metoder og at beskrive deres funktion, samt deres parametre og returværdier. Vi har valgt at beskrive de mest relevante metoder i vores API for at give andre udviklere muligheden for at kunne forstå og bruge vores interfaces i deres egne konstruktioner. Vi har også valgt at

skrive denne Java Doc på engelsk for at gøre den mere international, da koden er skrevet meget på dansk, så er det jo svært for udenlandske udviklere at forstå vores kode.

jUnit og brugertest (Martin)

Vi har valgt at lave jUnit tests på de mest vitale metoder af vores program, hvilket blandt andet var bestil kørsels metoderne. Vi kørte disse igennem med jUnit hvilket er et indbygget plugin i Eclipse, og vi lavede i forvejen en testcase med de mulige kombinationer vi kunne komme i tanke om at metoden kunne blive brugt som. De variationer som var begrænset grundet valideringer og eller drop-down valgmuligheder testede vi ikke, da programmet aldrig burde kunne komme til disse variationer. Rent brugertestmæssigt testede vi hele programmet igennem 2 gange med nogle forskellige personer, og der blev blot skrevet noter ned om ting de gerne ville have lavet om, hvilket ikke var særligt meget. Disse noter kan findes i bilag 11 og disse blev hurtigst muligt ændret / fikset efter vi fik dem at vide, i hvert fald de ændringer vi også selv følte var relevante. Disse tests var med til at kvalitetssikre vores program, samt hjalp de med at opnå vores ikke funktionelle krav såsom at programmet skulle være let at bruge.

Metrics gennemgang (Martin)

Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Number of Parameters (avg/max per		0,796	1,175	11	/FlexMartinSteffen/FlexMartinSteffen/src/l...	angivInformationer
Number of Static Attributes (avg/max	27	0,794	2,643	13	/FlexMartinSteffen/FlexMartinSteffen/src/p...	
Efferent Coupling (avg/max per pack		5,6	4,128	12	/FlexMartinSteffen/FlexMartinSteffen/src/p...	
Specialization Index (avg/max per typ		0	0	0	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Number of Classes (avg/max per pac	34	6,8	3,311	13	/FlexMartinSteffen/FlexMartinSteffen/src/p...	
Number of Attributes (avg/max per t	245	7,206	10,366	37	/FlexMartinSteffen/FlexMartinSteffen/src/p...	
Abstractness (avg/max per packageFi		0,235	0,205	0,5	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Normalized Distance (avg/max per p		0,185	0,212	0,5	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Number of Static Methods (avg/max	2	0,059	0,235	1	/FlexMartinSteffen/FlexMartinSteffen/src/l...	
Number of Interfaces (avg/max per p	13	2,6	2,332	6	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Total Lines of Code	3576					
Weighted methods per Class (avg/ma	390	11,471	10,239	40	/FlexMartinSteffen/FlexMartinSteffen/src/l...	
Number of Methods (avg/max per typ	228	6,706	7,351	36	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Depth of Inheritance Tree (avg/max p		1,147	0,429	3	/FlexMartinSteffen/FlexMartinSteffen/src/l...	
Number of Packages	5					
Instability (avg/max per packageFrag		0,581	0,386	1	/FlexMartinSteffen/FlexMartinSteffen/src/p...	
McCabe Cyclomatic Complexity (avg/		1,696	1,627	11	/FlexMartinSteffen/FlexMartinSteffen/src/l...	validerInformationer
Nested Block Depth (avg/max per me		1,37	0,69	5	/FlexMartinSteffen/FlexMartinSteffen/src/p...	haandterUdregn
Lack of Cohesion of Methods (avg/m		0,457	0,441	1,167	/FlexMartinSteffen/FlexMartinSteffen/src/p...	
Method Lines of Code (avg/max per	1751	7,613	11,548	74	/FlexMartinSteffen/FlexMartinSteffen/src/p...	haandterUdregn
Number of Overridden Methods (avg	0	0	0	0	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Afferent Coupling (avg/max per pack		8,2	8,818	22	/FlexMartinSteffen/FlexMartinSteffen/src/d...	
Number of Children (avg/max per typ	0	0	0	0	/FlexMartinSteffen/FlexMartinSteffen/src/d...	

Figur 9 Metrics analyse af vores program

På figur 9 kan en metrics analyse ses af vores program. Metrics er et plugin til Eclipse der kan analysere ens kode for blandt andet kompleksitet. Vores kode ser nogenlunde ud mht. til kompleksitet (se McCabe Cyclomatic Complexity) dog er der en enkelt metode som der bryder vores gennemsnit, og dermed gør den

rød i analysen. Denne ene metode er blot validerInformationer, og den er for kompleks da den modtager for mange forskellige informationer den skal validere. Det ville dog ikke være bedre at dele denne op, da overblikket over koden ville blive væk, så derfor gjorde vi ikke noget ved denne ene metode. Udover denne kompleksitet modtager vores angivInformationer metode en tand for mange parametre med sig, men igen denne ville være uhensigtsmæssigt at ændre, da der ellers ville blive mistet overblik over klassen, i den allerede store Controller klasse. Den modtager så mange informationer, da den er den primære kobling imellem vores bestillings brugergrænseflade og vores logik lag. Alt i alt giver metrics os et godt indblik i kvalitetssikring af vores kode, og skulle vi arbejde mere med kvaliteten af koden ville det primært være disse to punkter.

Review af vores artefakter

Reviewet af vores artefakter skete på baggrund af at vi gerne ville kvalitetssikre selve systemudviklingen i vores program, så derfor valgte vi at reviewe hele usecase 1's artefakter, udover at kigge vores andre artefakter uformelt igennem. Vi kom ikke frem til nogle store programmæssige/logikmæssige ændringer, men vi rettede sporbarhedsfejl i de forskellige artefakter. Review notaterne kan ses i bilag 12.

Konklusion

I dette projekt har vi fået arbejdet ud fra vores problemformulering, visionsdokument og kravspecifikationer, og har vist at vi har endt med at lave et professionelt IT-program der dækker over hvad vi originalt havde sat ud til at lave med det. Vi startede med at overveje specificerede krav, og kom rundt omkring disse i form af vores visionsdokument, som satte grundstenene samt det endelige mål for vores projekt. Udover det har vi lavet en dokumentation af vores kravspecifikationer og vi kan konkludere at vi har nået disse krav samt levet op til vores visionsdokument. Dette kan vi konkludere ud fra vores færdigudviklede program, der har de nødvendige funktioner, samt brugertests der har vist en opfyldelse af de ikke-funktionelle krav, bl.a. at brugergrænsefladen skulle være let forståelig og intuitivt, og at brugeren ikke skulle opleve længere ventetider i henhold til at programmet tog for lang tid om at arbejde ud fra brugerens inputs.

Vi kan også bevise at vores endelige program er et professionelt IT-program, i den forstand at vi har kvalitetssikret programmet på diverse måder, såsom jUnit Test af udvalgte klasser, dokumentation af programmet i form af systemudviklings artefakter og kvalitetssikring af disse artefakter i form af reviews.

Hvis vi skulle videreudvikle programmet kunne der være lavet mere specifikke fejlbeskeder bl.a., samt tilføjelse af observer kald til nogle enkelte steder hvor det er nice-to-have. Udover det er der nogle rent

kodemæssige ting vi kunne have ændret for æstetisk at få vores klasser til at være pænere opsat så det er nemmere at danne sig et overblik. Men alt i alt fik vi gennemført vores projekt på en hensigtsmæssig måde med de specificerede succeskrav.