Private Banking Simuleringsværktøj

Klaus Christiansen Jes Conradsen
Christian Birkholm Clausen

3. April 2006

Indhold

Kapitel 1

Opgavestart

1.1 Indledning

Formålet med denne rapport er at beskrive og dokumentere vores projektforløb i forbindelse med hovedopgaven, på 5. semester af datamatikeruddannelsen på Niels Brock Business College i København. Opgaven er skrevet i samarbejde med EIK Bank, en investeringsbank med oprindelse i Island og Færøerne. Rapporten er skrevet sideløbende med projektet, i tidsrummet 23. januar til 3. april 2006, og med EIK Bank og Niels Brock som målgruppe.

Projektet er lavet med udgangspunkt i den, af EIK Bank, stillede opgave, der omhandler udviklingen af et værktøj til simulering og håndtering af rådgivningen af privat økonomi. Denne rådgivning, som primært drejer sig om formuerådgivning og -pleje, kaldes Private Banking og er et af bankens kerneområder, hvor der fokuseres på kunder med likvider for minimum 1 million kroner.

Rapporten beskriver de problemstillinger, overvejelser og beslutninger vi mødte i forbindelse med analyseringen af problemområdet, designet og implementeringen af applikationen, samt de projektstyrings- og rapportmæssige aspekter. Man vil som læser, med andre ord, støde på emner som LaTeX, XP, C# .Net 2.0, Prototyping, Microsoft Access og Crystal Reports. Emner som alle var en stor del af vores tid med projektet.

God læselyst!

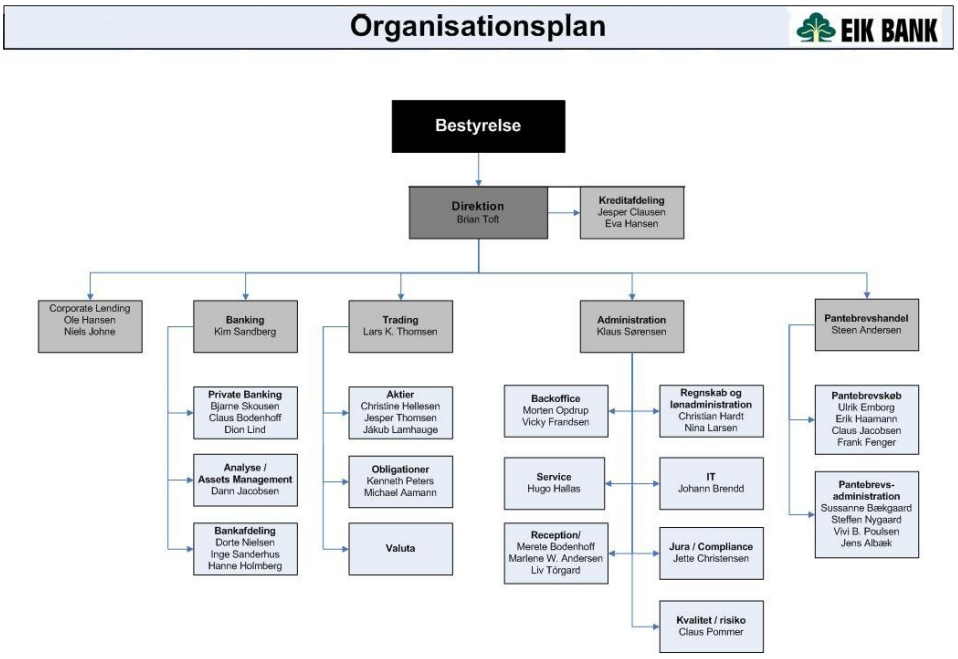
1.2 Virksomhedsprofil

Banken åbnede i januar 2001, som repræsentationskontor for den islandske Kaupthing Bank. Her var banken 75% ejet af Kaupthing og 25% af Føroya Sparikassi. Føroya Sparikassi er Færøernes første og største pengeinstitut og blev stiftet i 1832. 1. januar 2005 opkøbte Føroya Sparikassi Kaupthings del af selskabet og det blev navngivet EIK Bank Danmark A/S. EIK er det færøske ord for det solide gamle egetræ, der i mange år har været fælles logo for de danske sparekasser.

Siden da har banken været i kraftig vækst, hvilket afspejles i halvårsresultatet for EIK Bank der steg fra 7,3 mio. kr. første halvår 2004 til 15,2 mio.kr. første halvår 2005. Balancen steg med 43% til 1,5 mia. kr. fra ultimo juni 2004 til ultimo juni 2005. Banken er medlem af Københavns Fondsbørs og leverer tjenesteydelser indenfor private banking, investeringsrådgivning, pantebrevshandel samt ejendoms- og projektfinansiering. Rådgiverne i banken har alle direkte kontakt til børsmarkedet. Banken har i dag 40 ansatte i dens ejendom på Nørre Farimagsgade 15 i København.

Bankens mål er langsigtede kunderelationer, i form af uvildig finansiel rådgivning.

I forbindelse med brugernes PC-kundskaber, kan medarbejderne generelt betegnes som brugere på normalt nivaeu.



Figur 1.1: Organisationsplan

1.3 Problemstilling

Det problemområde vores projekt ligger inden for, er de private kunder der søger rådgivning af deres formuer, og hører dermed ind under Private Banking.

Private Banking består af rådgivning af kunder og planlægning af deres økonomiske forhold. De økonomiske forhold deles op i indtægts-, formue- og pensionsforhold. Kunderne kan for eksempel få rådgivning om hvordan kundens økonomi ser ud i tilfælde af uarbejdsdygtighed, hvordan man bedst sikrer sig en optimal udnyttelse af kundens likvider, om disse likvider kan omstruktureres og så videre. Det primære problem består i at kunne vise kunderne en troværdig oversigt over og simulering af nuværende og fremtidige forhold. Man forsøger at sikre kunden bedst mulige økonomiske vilkår fremover, ud fra det grundlag kunden står med.

Problematikken med afdelingens nuværende arbejdsgang drejer sig hovedsageligt om de værktøjer rådgiverne har til rådighed. Bankens Private Banking afdeling benytter i dag Excel regneark, hvor alle tal og forhold indtastes, behandles via formler og derefter printes ud til kunden. Et eksempel på det nuværende system kan ses i bilagene, bilag 10.2, side 112. Indtastningen af kundens data sker ud fra det materiale kunden kommer med, som er på papirform. Dette materiale drejer sig typisk om depotoversigter, selvangivelser, årsopgørelser med mere. Sammensætningen af materialet er endvidere forskelligt fra kunde til kunde, hvormed registreringen af oplysningerne ikke kan gøres til en automatiseret process. Rådgiveren skal altså holde styr på papirene samtidig med at de relevante data observeres og tastes ind. Derfor er det vigtigt at indtastningsdelen på skærmen er så nem at have med at gøre som muligt, hvilket man ikke kan sige at regneark er. I regneark har man adgang til alle felter, man kan komme til at skrive i det forkerte felt og slette data der ikke skulle være slettet. Endvidere skal man selv sørge for at slette tidligere data fra hvert felt, da den skabelon regnearket kopieres fra er en tidligere sag. Afdelingens rådgivere ønsker sig i dette regi et værktøj der kan lette indtastningen af kundens data.

Endvidere må muligheden for at lave præsentable udprint, baseret på regneark, siges at være begrænset. Dette beskrives af afdelingens rådgiverer som et andet af hovedproblemerne, at *"man er ikke nødvendigvis stolt af det resulterende materiale man giver til kunden"*. Problemet er at servere en økonomisk oversigt for kunden, på en mere overskuelig og præsentabel måde.

Et andet problem består i at kundernes data i Private Banking afdelingens system ikke er centraliseret. Sagerne gemmes på et fælles netværksdrev. Alle har samme adgangsrettighed til denn mappe. Hver rådgiver holder yderligere styr på sine kunder i personlige adressekartoteker, da en kunde normalt tilknyttes én og samme rådgiver, som følger kunden så længe kunden er tilknyttet banken. Dette kan give problemer hvis en rådgiver bliver

syg, og de andre derfor ikke har adgang til disse data.

1.4 Formål med systemet

Dette afsnit opsummerer hovedpunkterne fra problemstillingen og anskuer formålet med det nye system fra EIK Banks side.

Det overordnede formål med det nye system er at gøre det nemmere for den private kunde at få et overblik over sin økonomi, på en professionel, indbydende og præsentabel måde. Banken er ikke tilfreds med deres nuværende værktøj som det primære arbejdsredskab og ser dette som et problem.

Yderligere er det et mål at lette bearbejdningen af data for EIK Banks Private Banking rådgivere. Det er relativt besværligt og usikkert at arbejde med Excel på indtastnings- og beregningsniveau, hvor en applikation med windows-form brugergrænseflade tilbyder langt bedre håndtering og sikkerhed. Med sikkerhed tænkes blandt andet på at man ikke kan komme til at skrive eller slette elementer der ikke er adgang til, som man kan i et regneark.

Det er endvidere et mål at samle rådgiverens værktøjer i én applikation. Disse værktøjer indbefatter de nuværende Excelark og dokumenter, fremskrivningsskabeloner, adressekartoteker og tabelopslag på diverse offentlige instansers hjemmesider.

1.5 Problemformulering

Ud fra de betragtninger vi har gjort om det eksisterende system, hvor EIK Bank anvender Excel regneark til simulering af private kunders økonomi, ses det tydeligt at et nyt system vil kunne bidrage med væsentlige forbedringer. Disse forbedringer vil til dels komme rådgiveren og dennes arbejdsgang til gode og dels vil de komme kunden til gode ved et hurtigere og bedre overblik, i form af et mere professionelt udprint. Systemet skal ikke lave væsentligt om på de processer der ligger i den nuværende arbejdsgang, men gøre det lettere at udføre arbejdet. For at vi kan finde ud af hvordan det nye system skal konstrueres, er det nødvendigt at analysere en række forhold, hvoraf problemformuleringen kan defineres:

EIK Bank har et krav om at systemet skal være brugervenligt, kunne udskrive professionelt udseende rapporter til kunden og skabe gennemskuelighed og overskuelighed af kundens økonomi. Hvordan kan vi opfylde dette krav ved inddragelse af eksperimentielle systemudviklingsmetoder og teori om brugervenlighed?

Et andet krav siger at systemet skal være modulært og udvidelsesvenligt. Kan vi ved brug af design patterns opfylde dette krav, og i så fald med hvilke design patterns?

Kapitel 2

Metodologi

2.1 Metodologi

Med henblik på den grundlæggende metodologi handler dette afsnit om de filosofier og principper vi har arbejdet efter og hvordan de er blevet anvendt.

For at beskrive en metodologi kigger vi på hvordan projektet er nedbrudt i faser, hvilke opgaver der skal udføres i hver fase og hvilket output der skal produceres. Yderligere ses der på hvem der involveres i faserne og opgaverne, hvornår de udføres og hvordan projektet skal styres. Man kan, hvis man finder det nødvendigt, også se på hvilke forhold opgaverne udføres under og hvilke hjælpeværktøjer man kan tage i brug. Se kilde [1].

I projektets første uge fokuserede vi på at blive enige om hvilke principper vi ville arbejde efter og have som styringsprincipper.

2.1.1 Filosofier

Vi blev hurtigt enige om følgende, helt grundlæggende, principper eller filosofier om man vil:

- Vi vil forsøge at arbejde ud fra en risiko-drevet tankegang, forstået på den måde, at den næste opgave vi skal løse skal være den opgave der er størst konsekvens ved ikke at få løst. Hermed er vi hele tiden opmærksomme på "at knække den hårdeste nød først".

- Vi har et princip om at vi skal dele rollerne mellem os på en sådan måde at de roterer blandt os. Dette er for at mindske risikoen for at vi sidder fast i gamle rutiner og arbejdsformer. Nogle er for eksempel mere vant til at styre møder og prototypesessioner end andre og andre er mere vant til at holde sig i baggrunden og observere. Nogle er gode til at kode, hvor andre er bedre til det systemudviklingsmæssige aspekt. Det skal ikke være fastlagt at dem der er vant til en opgave, dermed er selvskrevne til at udføre opgaven næste gang. Vi vil derudover også

på denne måde hver i sær forsøge at udvikle os på andre områder end vores kerneområder. Der er dog også et paradoks med denne rotation, på den ene side skal alle være med til alt, på den anden side kan der spares tid og udvikles mere effektivt, hvis folk laver det de er bedst til. Her kan følelsen af tidspres, være den faktor der gør at man afviger fra sit princip.

- Vi inddrager elementer fra eksperimentiel systemudvikling. Her tænkes specielt på prototyping. Vi vil endvidere anvende elementer fra de agile metoder, hvor for eksempel det daglige morgenmøde er inspireret af den agile udviklingsmetode Scrum. De agile metoder er netop kendetegnet ved at være fleksible og kunne tilpasse sig forløbet, ved hyppig opfølgning på processen.

- I forbindelse med kravindsamling vil vi gøre brug at prototyping, som et 'RAD'[1] moment. Vi vil primært gøre brug at den papirbaserede Lo-Fi [2] prototype, kombineret med video- og lydoptagelse af sessionerne.

- Aktiv brugerdeltagelse er væsentlig for os, hvorfor vi afholder en ugentlig briefing for de, i projektet, involverede personer fra EIK Bank. Udover dette inddrager vi brugerne i udviklingen, her tænkes på prototypesessioner, kravindsamling, møder med mere.

- Vi anvender ikke UP[3]. Hermed menes, at vi ikke deler vores projektforløb op i inception, elaboration, construction og transition, i en fastlagt iterativ process. I forbindelse med dokumentation over for opgaven arbejdes der ikke efter en dokumentdrevet tankegang (dette er med inspiration fra XP). Vi har, i opgaven, ikke noget ønske om at lade os drukne i dokumentation. I forhold til selve systemet må vi dog stille vores eget ønske om let dokumentgang til side og lave den grad af dokumentation som virksomheden ønsker. Vi vil altså tage elementer fra UP i brug når vi føler det bliver relevant.

- Vi vil, og skal, stille så mange "dumme" spørgsmål som muligt. Ikke for at stille dumme spørgsmål, men for at få de bedst mulige forklaringer på de ting vi spørger om. Problemet er at vi, såvel som de involverede medarbejdere fra EIK Bank, er fagspecialister på hver vores område. Derfor kan det være vanskelig at kommunikere med hinanden på en sådan måde at alle parter forstår hinanden. Derfor vil vi hellere spørge ind til et problem en gang for meget, end en gang for lidt.

[1] Rapid Application Development
[2] Low Fidelity
[3] Unified Process

2.1.2 Arbejdsform

Det overordnede framework for projektet er inspireret af spiralmodellen og dens iterative adfærd. Vi nedbryder dermed projektforløbet i faser. Herunder er det fastlagt, at vi har korte iterationer på 1 uges varighed.

- For hver iteration skal der løses fastlagte opgaver:
  - Dagen starter med at afholde et internt morgenmøde, hvor der bliver gjort status og lavet dagsorden for hvad der skal laves den pågældende dag, se Bilag 10.3 på side 113. Dette er efter forbillede af XP og Scrum. Dagsordenen nedskrives til hvert møde og gemmes.
  - Når dagen er omme skrives der dagbog, hvor der bliver noteret hvad der er lavet og sket, se Bilag 10.4.
  - Der laves en ugentlig planlægning af næste iteration. Dette vil sige at der hver fredag bliver lavet en plan for næste uges forløb og denne uges mål. Det ugentlig mål resulterer typisk i færdiggørelsen af et eller flere af projektets delmål, altså et delsystem. Hver iteration løber fra fredag til torsdag, hvormed deadlinen for ugens mål er fredag morgen. Den ugentlige plan nedskrives og gemmes. I dette planlægningsforløb opklares hvad der i kommende iteration er den "hårdeste nød". Dette gøres internt i gruppen.
  - Der afholdes et ugentligt informationsmøde med vores kontaktpersoner fra EIK Bank. Dette er for at vise fremskridt, engagement, åbenhed og for at bevare interessen omkring vores arbejde. Endvidere er vi interessede i løbende at sikre os at vi og EIK Bank er enige om det arbejde der bliver udført. Efter møderne skrives der referater, som vi bruger til at behandle indtryk fra og resultater af møderne. Mødet er stilet mod vores to kontaktpersoner fra private banken afdelingen, samt to medarbejdere fra IT-afdelingen.
  - Der skal bruges tid på at skrive rapport mindst én gang om ugen. Som udgangspunkt sker dette hver fredag, hvormed fredag bliver en "skrive-dag".

2.1.3 Vore erfaringer med værktøjer og metodologien

Noget af det første vi erfarede i forbindelse med vores principper, var at mål og planer ikke har nogen effekt hvis der ikke er konsekvens bag ordene. Det er ikke hensigtsmæssigt bare at rykke en deadline eller lade være med at opfylde et mål, hvilket kan ske hvis der ikke er nogen konsekvens ved at gøre det. Dermed er det vigtigt at få defineret sine mål præcist og at målene holdes inden for så realistiske rammer som muligt. Ved at være opmærksom

på dette får vi også bedre mulighed for at få success med at planlægge os ud af eventuelle problemer.

Et af de første mål, der kunne opfattes som en hård nød, var at bestemme hvilken database der skulle anvendes og hvor denne database skulle ligge. Læs mere om denne problematik i 5.2, på side 45. For at få svar på dette spørgsmål skrev vi en mail, der blev sendt videre til bankens datacentral. Problemet var dog at vi ikke havde talt om hvad vi skulle gøre hvis vi ikke fik svar på mailen inden vores deadline udløb. Dette fik vi ikke og dermed stod vi med et problem. Efter en del overvejelser blev det vedtaget at holde os til planen, i henhold til ovenstående afsnit. Dette gjorde vi ved at tage en beslutnig om at arbejde videre med en løsning som vi vidste var mulig, nemlig at anvende en Access database, fremfor at risikere at spilde tid på en uholdbar løsning. Vores tanker og ønske om at arbejde med en mere avanceret databaseløsning blev dermed droppet og vi havde nu nået vores mål.

De daglige rutiner, her tænkes specielt på morgenmødet, har vist sig at have en god effekt på arbejdsgangen, på flere måder. For det første hjælper morgenmødet til at sikre at alle gruppemedlemmer ved hvad der er lavet, hvad der skal laves, hvad der foregår og hvem der laver hvad. Mødet er yderligere en god måde at sikre at der bliver taget hånd om vigtigste problemer, i henhold til ugeplanlægningen. Det er altså et et godt værktøj til at holde fokus på det arbejde der skal udføres og til at holde styr på hvad der er lavet. Her er dagbogen også et hjælpeværktøj som vi, i samspil med morgenmødet, har haft gavn af. Dagbogen er knapt så formel og kræver ikke nødvendigvis et decideret møde internt i gruppen, men gemmes og giver mulighed for tilbageblik og status, se Bilag side 126.

Den ugentlige planlægning af næste iteration har været en effektiv måde til at identificere de mest presserende problemer og til at koordinere den tid der har været til rådighed. Med denne planlægning har vi, som før skrevet, haft fokus på at identificere de "hårdeste nødder". De sidste to uger af projektet, har vi dog ikke haft overordnede mål for udviklingen, da ugerne har været reserveret til rapportskrivning. Rapportskrivning mener vi umiddelbart ikke kan planlægges og koordineres så lang tid i forvejen, hvorfor der ikke udført ugeplanlægning i denne fase.

Iterationens længde og varighed, fra fredag til og med torsdag, har været et punkt der tog noget længere tid at beslutte sig for. Et argument mod den valgte varighed, har været at vi reelt kun har haft 4 dage til udviklingen i hver iteration, i og med at fredag, som iterationens første dag, hovedsageligt er blevet brugt til fredagsbriefing, planlægning af iterationen samt rapportskrivning. Man skal dog huske på at disse ting også er vigtige at tage sig tid til. Vi har altså haft mulighed for, via fredagens aktiviteter, at planlægge, koordinere og vurdere hvilke mål der skal sættes for iterationen. Dette har virket ganske godt og vi føler selv at have sat relativt realistiske mål.

For at "holde kontakten" med virkomheden og dens medarbejdere har

vi, som skrevet, afholdt et briefingsmøde hver fredag kl 09:30. Vores agenda, til dette møde, har været at forklare hvad vi har lavet i den forgangne uge og hvad planen er for den kommende uge. Der er blevet forberedt en invitation til hvert møde, hvor dagsordenen er blevet præsenteret på punktform. Denne invitation er blevet sendt ud til de relevante personer dagen inden. Dette har vi til dels gjort for at give medarbejderne en mulighed for at bedømme om mødet kunne være interessant for dem eller ej og dels for at give medarbejderne mulighed for at forberede sig til mødet. Vi føler at vi har opnået en god kontakt med bankens medarbejdere via briefingsmøderne og vi har uden tvivl fået sat mange ting på plads og imødekommet tvivl og misforståelser. For eksempel havde vi på et tidspunkt taget en afgrænsningsmæssig beslutning, hvor vi var af den opfattelse at fremskrivningerne var vigtigere end udprintsfunktionaliteten. Dette fremlagde vi til mødet og det viste sig at EIK Bank ikke var enige i denne prioritering. Hermed undgik vi at arbejde videre ud af "et forkert spor" og vi valgte, som resultat af mødet, at lytte til EIK Banks mening og fokusere på udprintningen.

Et af de første projektstyringsmæssige tiltag vi tog, var at udvikle en tidslinie over projektforløbet på kalenderform. Denne tidsplan fylder én A4 side og er inddelt i 10 uger, fra d. 30. januar til d. 3. april. Det gav os et tidligt overblik over hvor mange iterationer vi har haft til rådighed og hvordan tiden kunne prioriteres. Ud fra dette overblik blev vi enige om følgende inddeling:

- Uge 4: Den første uge af projektet går med planlægning, indledende tanker og installation på kontoret.
- Uge 5, 6, 7, 8, 9, 10 og 11: De 7 uger/iterationer vi har til analyse, design og implementering.
- Uge 12 og 13: Rapportskrivning.

I forbindelse med den mere praktiske del af projektstyringen, har vi fra starten anvendt et system til versionsstyring. Systemet hedder SVN[4] og tager sig af håndtering af dokumenter, kode og andet, med andre ord alt vores materiale. Systemet fungerer som en server- og klient løsning, hvor serveren holder og styrer repositoriet hvor data ligger. Denne Subversion serverapplikation kører på en ekstern Linux server. Via Windows-klienten Tortoise SVN kan man tilføje nyt materiale, redigere i det eksisterende materiale og hvis noget skulle gå galt, genskabe data fra tidligere revisioner. Vi har haft adgang til projektets data derhjemme, nøjagtig som vi har haft det på vores computere på EIK Bank. Systemet holder styr på hvem der har lavet hvad og hvornår det er blevet lavet. Data slettes aldrig fra repositoriet selvom det bliver slettet via klienten, det bliver bare fjernet fra den aktive del af repositoriet, hvorfor man altid har mulighed for at finde data frem igen.

[4]Subversion

Kapitel 3

Kravindsamling

3.1 Prototyping

3.1.1 Vores brug af prototyping

For at skabe en, mellem udvikler og bruger, fælles forståelse af systemet, kan det være ganske frugtbart og forholdsvist omkostningsfrit at bygge en prototype og bruge denne som en præ-udviklingsplatform. Der findes forskellige former for prototyper, som har det til fælles at de er meget dynamiske og relativt lette samt omkostningsfrie at lave ændringer i. Hermed er det ikke altafgørende at prototypen er komplet, fra første gang den anvendes i en session mellem udviklere og brugere. Det vigtige er, at man kommer hurtigt i gang med at specificere krav, snakke layout, undersøge workflow med mere. Prototyping er altså et eksperimenterende element i vores systemudviklingsmetode, som står i kontrast til den typisk mere dokument- og analysedrevne og traditionelle form for systemudvikling, såsom Unified Process.

Det vigtigste aspekt i forbindelse med prototyping er fastsættelsen af krav til systemet. I denne henseende er prototyping lige så meget et hjælpeværktøj for kunden/brugeren, som for udviklerne, da kunden langt fra altid selv er klar over hvilke krav de har til et fremtidigt system. Kunden får et tidligt indblik i systemet og dermed længere tid til at tænke over hvad de vil have systemet til at kunne.

Netop aspektet med at man hurtigt kommer igang med at visualisere systemet og fastlægge krav, ligger til grund for vores valg af prototypingteknik, nemlig Lo-Fi. prototyping teknikken. Lo-Fi er en papirbaseret prototype der repræsenterer et skærmbillede af det fremtidige system. Under sessionen uddeles der roller til udviklerne, nemlig facilitator, 'computer' og observatør. Det er udviklernes opgave at afholde og styre sessionen, hvor kunden/brugeren bliver præsenteret for prototypen og en række scenarier der skal gennemgås. Kunden/brugeren kan nu udføre den stillede opgave ved at 'lege' at han anvender systemet, hvor 'computeren' sørger for at systemet udfører det kunden/brugeren gør. Vi har endvidere valgt at anvende

Midi-Fi[1] prototyping, i form af video-, billeder og lydoptagelser af vores prototype sessioner, til dels for at støtte observatøren og til dels for at dokumentere vores process, se figur 4.4 på side 32, figur 4.5 på side 32 og figur 4.6 på side 33.

Prototyping er, som før nævnt, et eksperimenterende værktøj, hvorfor det er vigtigt for udviklerne at have en god forståelse for og et godt kendskab til prototyping. Der er mange situationer hvor manglende styring og mangelfuld forberedelse væsentligt kan mindske det udbytte man får af en prototype. Ved at undersøge teorien omkring prototyping, inden man anvender teknikken, opnår man det bedste resultat. Vi har som studerende på datamatikeruddannelsen haft prototyping og eksperimentiel systemudvikling som emne på fjerde semester, hvor vi udover at studere teorien, lavede prototyping studiegrupperne imellem. Det er ud fra denne erfaring at vi har anvendt prototyping i forbindelse med vores hovedopgave projekt hos EIK Bank.

Som udgangspunkt har vi lagt fokus på indtastningsdelen af systemet og har derfor udviklet prototyper til denne. Vi har startet med at lægge fokus på indtastingsdelen, da denne del er vital for resten af systemet. Den var det mest logiske sted at begynde, da ingen af systemets resterende delsystemer, fremskrivninger og udprintning, kan eksistere uden en fungerende indtastningsdel. Yderligere udtrykte EIK Bank ønske om et mere afgrænset og velfungerende produkt, hvorfor vi ikke ønskede at gabe over for stor en mundfuld.

Til at simulere de opgaver en bruger kan komme ud for, har vi fået materiale i form af en rigtig sag fra private banking. Ud fra dette materiale kunne vi derfor skabe troværdige scenarier.

3.1.2 Prototype 1

Vi har arbejdet med denne prototype i projektets anden uge/iteration.

Formål

- At få et tidligt indblik i EIK Banks arbejdsgange i forbindelse med indtastning af en kundens stamdata og økonomiske forhold.
- At få afprøvet vores visioner omkring systemets virkemåde.
- At få indsamlet krav og specificeret krav til systemet.
- At få øvelse i at afholde prototyping-sessioner.

[1] Midi-Fidelity.

Setup

Vi valgte at bygge vores første prototype på basis af en, i gruppen, intern brainstorming. Her brugte hvert medlem af gruppen en halv time på at forme sin egen version af systemet. Ud fra de resulterende skitser evaluerede vi vores ideer og tanker og samlede derfra det bedste fra hver udgave, til et samlet udkast. Dermed blev prototype 1 til. Se vedlagte billede under bilag, reference.

Der blev afholdt en session med vores kontaktpersoner fra private banking, som begge er rådgivere, hvor setup'et var ens i form af samme prototype og scenarier. Grunden til at vi holder en session med hver rådgiver er, at vi gerne vil have den enkelte rådgivers personlige mening og opfattelse. Vi var ikke interesserede i at rådgiverne kommer til at farve hinandens udmeldinger eller på anden måde påvirker hinanden. Vi anvendte videooptagelse af sessionerne, for at fastholde dokumentationen. Gruppemedlemmerne var tildelt roller, i form af facilitator, 'computer' og observatør. Til sessionerne forberedte vi scenarier til brugeren, som havde til formål at simulere situationer brugeren kan komme ud for i den nuværende arbejdsgang. Den første session var planlagt til at vare 1 time, hvor brugeren skulle gennemgå de opstillede scenarier, nemlig 4 styk:

1. *Indtast kundens stamdata ud fra det udleverede materiale.*
2. *Indtast kundens indtægtsforhold.*
3. *Indtast kundens formueforhold.*
4. *Indtast kundens pensionsforhold.*

Gennemførelse

Session 1

Vi startede med at præsentere os selv og vores roller. Derefter forklarede vi hvad vi havde planlagt til sessionen og hvad vi gerne ville nå frem til. Derudover spurgte vi om det var i orden at vi optog forløbet på video, dette fik vi en accept af. Sessionen blev gennemført ved gennemgang af scenarierne, dog tog den længere tid end planlagt, vi overskred tidsplanen med cirka et kvarter. Der blev afrundet med evaluerende spørgsmål og indtryk fra både os og rådgiveren.

Session 2

Denne session var, som session 1, planlagt til at vare 1 time. Vi ville her forsøge at være bedre til at overholde tidsplanen. Vi forsøgte at bruge lidt mere tid på at introducere og forklare forløbet, opdelingen af vores roller under sessionen og hvad det i det hele taget gik ud på. Vi fik igen accept

af brugen til at videooptage. Scenarierne blev gennemgået, men det lykkedes desværre ikke at holde sessionen indenfor den planlagte tidsramme. Sessionen blev afsluttet med en evaluering af forløbet.

Samlet evaluering

Vi fandt ud af hvor vigtigt det er at tage sig tid til at få forklaret hvad formålet med en session er, hvad den indebærer, hvad rollerne indebærer og hvordan brugeren kan interagere med systemet. For eksempel burde vi have brugt flere kræfter på at forklare, at 'computeren' udelukkende reagerer på sessionsdeltagerens input og ikke gør andet end det. Dette punkt må gerne gå lidt langsomt. Vi var ikke alt for gode til dette, hvilket er en erfaring vi vil tage med os til næste session.

Vores brugere er rådgivere af natur. Hermed menes at de er vant til at styre samtaler og dialog. Dette menes ikke som en kritik, men som en konstatering af at facilitator skal være ekstra opmærksom på at styre sessionen, hvilket vi oplevede var svært. 'Computeren' var generelt ikke opmærksom på at melde fejl når brugeren udførte noget ulovligt, hvilket skaber forvirring i situationen. Denne forvirring udmynter sig i misforståelser af hvad systemet kan og man risikerer at brugeren tager styringen. Det er en øvelse at overholde de roller man har, hvilket ikke altid lykkedes lige godt selvom vi havde planlagt dem godt. Da vi har et princip om job-rotation, skiftede rollerne fra session 1 til session 2. Dette betyder også, at man ikke kan specialisere sig så meget i en rolle, og dermed må forvente at det tager længere tid før samarbejdet kommer til at køre som ønsket.

Gennem evaluering af vore egne indsatser, har vi forsøgt at blive bedre til at overholde og respektere de roller der er tildelt. Det er for eksempel ikke hensigtsmæssigt, som computer, at afbryde facilitator midt i en instruktion. Yderligere er det at lære og forstå hvordan folk hver især arbejder, og er forskellige, et punkt vi har arbejdet med.

Det kan være en stor hjælp for brugeren at lave et forklaringsark, hvor symbolerne (knapper, radioknapper, tekstfelter, osv.) på prototypen er beskrevet. Dette forklaringsark skal ligge fremme, til adgang for brugeren. Hermed har brugeren hele tiden muligheden for at bevare overblikket og ikke standse for meget op hvis han bliver i tvivl om hvad symbolerne betyder. Vi havde ikke et sådan forklaringsark under session 1, hvor vi observerede at brugeren enten havde glemt betydningen af eller ikke forstod symbolerne. Dette fik vi fulgt op på under session 2, med godt resultat.

Det er vigtigt at de skærmbilleder der ikke bruges gemmes for brugeren. Hvis ikke dette gøres risikerer brugeren at blive distraheret og springe i skærmbillederne, uden at bede computeren om det, hvilket mindsker graden af en realistisk simulering af en computers virkemåde. Yderligere forstyrrer det den plan vi har lagt for forløbet. På den anden side gav det os en forståelse af, at en stor grad af fleksibilitet i systemet er nødvendig. Samtidigt

fandt vi ud af at den navigeringsbar mellem skærmbillederne som vi havde lavet, mindskede graden af fleksibilitet da den var låst. Hermed menes at navigeringen foregik ét skærmbillede af gangen, uden mulighed for at springe et skærmbillede over. Dette påpegede rådgiveren i session 2 som et problem.

Generelt fik vi rigtig meget god information ud af sessionerne med prototype 1. Vi blev meget klogere på systemet og de nuværende arbejdsgange.

Vi vil gerne blive bedre til at estimere og forudsige varigheden af de sessioner vi afholder. Noget af det vi har gjort for at estimere er at vi hver især er kommet med vores bud på hvor lang tid en session vil tage og derefter ligge lidt til, for at forsøge på at være på den sikre side. Det har dog desværre, hver gang, vist sig at vi har lagt for lidt tid til. Vi fik god øvelse i prototyping og mange erfaringer med på vejen.

3.1.3 Prototype 2

Vi har arbejdet med prototype 2 i projektets tredie uge/iteration.

Formål

For at sikre kvalitet og for at få et optimalt udgangspunkt for at gå i gang med designfasen, har vi valgt at udvikle en forbedret version af prototypen, en 'færdig' prototype, om man vil. Denne prototype har de foreslåede forbedringer fra prototype 1 med, samt at vi valgte at lægge en del mere vægt på layout og udformning. Formålet med disse forbedringer var at gøre det nemmere at få indholdet til at være der på en pænere og mere præsentabel måde.

Til denne session har vi ønsket en samlet gennemgang, diskussion og dialog omkring forbedringer og eventuelle forværringer, og undladt at basere sessionen på scenarier. Diskussionen ønskede vi skulle foregå mellem rådgiverne, vi gav dog ikke udtryk for dette, da det helst skulle komme af sig selv. Hermed ønskede vi at få dem til at stille hinanden spørgsmål, som vi ikke selv ville kunne have tænkt på. Faremomentet er at de som fagfolk kan komme til at tale så specifikt at vi som udenforstående ikke kan forstå pointerne.

Målet med denne prototype kan opfattes som et kvalitetscheck af de allerede indsamlede data og en konfirmation af hvad vi er blevet enige om med kunden.

Formålet er altså at præcisere hvilke felter der skal være i indtastningsdelen i systemet.

Setup

Til prototypen brugte vi forskelligt farvet papir til at skelne mellem elementer, og skærmbilledearealet var dobbel så stort. Til denne prototype blev der afholdt en samlet session med de to medarbejdere der deltog i sessionerne

med prototype 1. Hvor vi i forbindelse med prototype 1 valgte at dokumentere på videoform, valgte vi i forbindelse med prototype 2 at nøjes med at optage lyd. Grunden til dette var at det var mere praktisk at nøjes med lydoptagelse på grund af det udstyr vi havde til rådighed, som ikke kunne optage video i mere end 3 minutter af gangen. Ved at lave lydoptagelse var vi fri for besværet med hele tiden at skulle tjekke udstyret. Derudover følte vi at det ikke var nødvendigt at dokumentere endnu en session med video, da det tager længere tid at behandle og bearbejde. Derudover tog vi et almindeligt foto af hver skærm i prototypen, for hver gang et skærmbillede blev debateret færdig og der blev enighed om hvad skærmbilledet skulle indeholde. Dette foto brugte vi som dokumentation af de forskellige skærmbilleders indhold. Der var igen tildelt roller til gruppemedlemmerne og der var afsat 1 time til denne session, se figur 4.4 på side 32, figur 4.5 på side 32 og figur 4.6 på side 33.

Gennemførelse

Vi startede med at præsentere de 2 brugere for hvad formålet med sessionen var og informerede om hvordan forløbet var planlagt. Vi forklarede at der var afsat 1 time til sessionen. Vi fik som svar hertil igen at vide at det tager den tid det tager og at vi ikke nødvendigvis skal stoppe før sessionen er fuldendt. Herefter blev der startet fra første skærmbillede og frem, hvor hvert skærmbillede blev gennemgået og vurderet. Yderligere blev de enkelte situationer, hvor indholdet af skærmbilledet ændrer sig, alt efter hvad man trykker på, vurderet. Hver færdig skærm blev, som planlagt, fotograferet.

Evaluering

Selvom sessionen desværre tog dobbelt så lang tid som planlagt, føler vi at den forløb godt. Den forløb velstruktureret, omend det kneb med koncentrationen til sidst på grund af det lange tidsforløb på 2 timer. Som erfaring har vi gjort os at 2 timer er for lang tid, der skal som det mindste være en god pause. Det er mere optimalt med 1 times varighed. Vi fik en rigtig god snak om arbejdsgangene og en konfirmation af hvad vi var kommet frem til op til dette punkt. Vi fik udarbejdet en udspecificeret 'færdig' prototype, som vi kan opfatte som en 'færdig' liste over hvad der skal være hvor i indtastningsdelen af systemet. Med hensyn til fordelingen af roller og det at overholde disse roller var sessionen rigtig god. Det virkede godt at vi nøjedes med at optage lyd denne gang. Dette aspekt virkede også mere afslappende og uformelt på brugerne, idet der ikke stod et kamera og filmede hver bevægelse. Under denne session var vi blevet bedre til at styre sessionen, brugerne og forløbet. Vi var mere opmærksomme på at styringen ikke må blive delt mellem brugere og udviklere. Vi fik også ros for at tage styringen.

3.1.4 Konklusion af vores brug af prototyping

Vi har fundet ud af hvor vigtigt det er at holde en stram styring af sessionerne og overholde de aftalte roller. 'Computeren' skal ikke gøre andet end at udføre det brugeren beder den om og melde fejl hvis det er nødvendigt. Det er facilitators opgave at sørge for at styre sessionen og brugeren, uden af give for meget hjælp. Brugeren må ikke få lov til at overtage styringen. Ligeledes er det vigtigt at lave en grundig introduktion til sessionen, hvor man tager sig tid til at forklare målet med sessionen, reglerne for sessionen med mere og hvor man sikrer sig at brugeren har forstået disse punkter.

Vi har både fået god prototype-teknisk erfaring og vi har nået målet med at få et tidligt indblik i arbejdsgange og krav.

Vi har fundet ud af at det er sværere end som så at spå om hvor lang tid en session vil komme til at tage. Sessionerne med prototype 1 tog begge væsentligt længere tid end beregnet, hvor sessionen med prototype 2 tog dobbelt så lang tid som planlagt. Dette er selvfølgelig ikke helt optimalt, både planlægningsmæssigt for os, såvel som for EIK Banks medarbejdere. Vi har dog været så priviligerede at EIK Banks medarbejdere har givet os den tid det måtte tage at gennemføre sessioner og møder, hvormed det i vores situation ikke har været et voldsomt problem at overskride tiden.

3.1.5 Konklusion i forhold til systemet

Vi fik opklaret en masse specifikke spørgsmål omkring hvilke felter, knapper, informationer med mere medarbejderne forventede i hvilke situationer. I denne henseende fandt vi ud af at mindre og smarte features, som for eksempel direkte links til www.skat.dk, skal prioriteres lavere end kernekravene og den basale funktionalitet i programmet, se afsnit 3.1 side 26.

Vi fandt ud af at der er stor forskel på hvordan medarbejderen griber det udleverede materiale an. En af medarbejderne sorterer først papirerne og deler materialet op efter person, hvor den anden tager papirene én for én og stort set i den rækkefølge de ligger i. Dette har betydning for os, i og med at systemet skal være så fleksibelt at det kan bruges af, og passer til, alle rådgiverne. Ydermere er der forskel på hvordan mennesker interagerer med computere. En medarbejder gik fint i spænd med vores rækkefølge af skærmbilleder, en anden ville gerne have mulighed for at springe i rækkefølgen. Dette var endnu et argument for at systemet skal være fleksibelt og ikke tilpasset en bestemt type bruger.

Vi fik et mere detaljeret indblik i workflowet, både i form af nuværende og fremtidige arbejdsgange. Dette er et meget vigtigt punkt, da vores system i første omgang ikke skal ændre på private banking afdelingens nuværende arbejdsgange, men primært fremme visualiteten og lette behandlingen af data. Dermed er det os der skal konstruere et system der passer til de eksisterende arbejdsgange. Det virkede som om at brugerne, under sessionerne,

forsøgte at gennemtvinge deres vante arbejdsgange i det nye system. Dette viser os at det kan være vanskeligt og måske endda urealistisk, at realisere nye arbejdsgange ved at indføre en nyt computersystem.

3.2 Kravliste

Vi ser det fremtidige system som bestående af to delsystemer. Et delsystem til indtastning og et delsystem til rapportering.

Skala: 1. prioritet betegnes som "Must have"
2. prioritet betegnes som "Could have"
3. prioritet betegnes som "Nice to have"

Til indtastningsdelen har vi fundet følgende krav:

Nr	Krav	Prioritet	Note
1	Systemet skal kunne printe (oversigter, grafer ol.)	1	I første omgang 'begrænset'.
2	Systemet og udprint skal have et professionelt look	1	Genbrug af officiel font, farver, logo, design stil osv.
3	Kunden ser måske programmet under indtastning og skal derfor kunne genkende applikationens stil på udprintet.	1	
4	Rapporten skal udstråle konservatisme, gennem overskulighed, gennemskuelighed og troværdighed.	1	
5	Der skal ikke printes logo.	1	Der bruges brevpapir.
6	Man skal kunne skifte database ved kun at modificere ét modul.	1	
7	Systemet skal kunne bruges af en person som er indøvet i Private Bankings papirgange, og som kan læse og skrive.	1	

Forsætter på næste side.

Nr	Krav	Prioritet	Note
8	Man skal kunne springe mellem de forskellige indtastningsskærme.	1	
9	Systemet skal ikke kræve at alle felter udfyldes.	1	Hvis en kunde ikke har nogle pensioner, så skal disse ikke tastes ind. = fleksibilitet.
10	Der skal være kontrol af typen der bliver indtastet i enkelte felter.	1	Telefon, Cpr må kun indeholde tal, og så skal der kun kunne indtastes tal.
11	Systemet tager sig af positive og negative tal.	1	Udgift = negativ, indtægt = positiv som default. Dette kan ændres ved at skrive minus foran.
12	Skærmene skal opdatere sig når noget nyt er tastet ind.	1	
13	Der skal være labels til indtastningsfelter som siger hvad der skal tastes ind.	1	
14	Man skal kunne oprette så mange emner i en kategori på en gang som muligt.	1	Eksempelvis skal muligheden for at taste renteindtægt, lønindkomst osv. ind på en indtægtsform være til stede uden at formen åbnes og lukkes efter hver indtastning. Med hensyn til aktier og obligationer har man tit mange af og disse skal derfor tastes ind i en datagrid.
Forsætter på næste side.			

Nr	Krav	Prioritet	Note
15	Emner der falder uden for kategori skal kunne tastes i et 'Andet' felt.	1	
16	Rapporten skal have så få sider som muligt.	1	Set i forhold til at alt skal kunne skrives ud og at det skal være overskueligt.
17	Systemet skal være brugervenligt.	1	
18	Opret, slet, læs og rediger en session.	1	
19	Opret, slet, læs og rediger en kunde.	1	
20	Opret, slet, læs og rediger et indtaegtsforhold.	1	
21	Opret, slet, læs og rediger et pensionsforhold.	2	
22	Opret, slet, læs og rediger et formueforhold.	2	
23	Søge på kunder.	2	
24	Systemet skal være nemt at lære.	2	
25	De forskellige lister af Aktiver, Passiver og Pensioner skal have kolonner med summer. En total og en for hver part i familien.	2	Er ændret fra 1 til 2, da det ikke er et primært krav.
26	Der skal være et simpelt notesystem. Dvs. en note der følger hele sagen.	2	Det har vist sig, pr. 24-02-06, at være en 2.
27	Alle indtastninger skal være brutto (altså før skat)	2	Ændret fra 1 til 2 da det ikke har det primærere fokusområde.
28	Under indtægtsforhold bør vi benytte FØR arbejdsmarkedsbidrag.	2	Ændret fra 1 til 2 da det ikke har det primærere fokusområde.
29	Efter 5-10 sager skal det tage den erfarne konsulent 15-30 min. at taste en kunde ind.	2	Det tager i dag 30-60 min.

Forsætter på næste side.

Nr	Krav	Prioritet	Note
30	Efter 5-10 sager skal det tage den nye konsulent 30-45 min. at taste en kundes data ind.	2	
31	Kontrol af indtastninger i felter.	3	Dette kunne være modulus 11 kontrol af CPR nr. Det kunne være at huske på at der er renteudgifter forbundet med et lån.
32	Der skal være et udvidet notesystem. Dvs. noter til hver post i en sag.	3	
33	Rådgiveren skal kunne anvende systemet uden for kontoret.	3	Det har vist sig, pr. 24-02-06, at være en 3'er.
34	Systemet skal kunne simulere fremskrivninger	3	Det har vist sig, pr. 24-02-06, at være en 3'er. Auto udregning. Dette afgrænser vi os helt fra.
35	Det skal være muligt at få et overblik over kunden og dennes personlige forhold, altså om kunden er selvstændig, bor i udlandet og så videre.	3	
36	Børn: Har de børn, hvor mange, deres navne, CPR numre og så videre. Er børnene fællesbørn eller ej? Er der tilknyttet nogen børneopsparing? Dette kan enten laves som punktform eller som en note.	3	Er ændret fra 1 til 3, dette kan skrives i notefeltet.

Forsætter på næste side.

Nr	Krav	Prioritet	Note
37	Børn skal ikke opfattes som en kunde.	3	
38	Er de lønmodtagere eller selvstændige (er det Aps. eller er det A/S), hvilke virksomheder de arbejder i. Dette kan enten lavet som punktform eller som en note.	3	Det skrive i en note.
39	Links til www.skat.dk og lignende så rådgivere kan tjekke huspriser og lignende.	3	
40	Det skal være synligt hvem der er kontaktperson for kunden.	3	
41	Adgangsbegrænsning og identificering via log-in	3	
42	Systemet er baseret på en cental database	pt. 3	I fremtiden 1.
43	Realisering af midler (skatte beregning)	3	
44	Man skal manuelt kunne ændre beskatningsprocenter.	3	
45	Systemet skal kunne printe alle data fra en session.	3	

Tabel 3.1: Kravliste.

Note: Når der skrives datoer ville det været smart hvis det er nemt at læse, men det skal også være muligt at sortere. For at undgå uoverensstemmelser med dato formater i koden og i databasen laver vi datoen som en streng. Det at kunne sortere er en feature vi afgrænser os fra. Generelt kan vi sige at vi afgrænser os fra 2. og 3. prioriteringer på grund af tiden.

Note: Krav 41 til og med 45 er ganske vagt formulerede og på ingen måde essentielle for at applikationen kan fungere. På nær krav 41, som vi har kendt siden begyndelsen af forløbet, er kravene fremkommet senere i udviklingsforløbet på baggrund af fremvisning af applikationen for EIK Banks rådgivere. Kravene er ikke nemme at kvantificere og man kan måske diskutere om de ikke nærmere har status af at være retningslinier for udviklingen af systemet. De er dog tilføjet vores kravsliste, idet vi vurderer at vigtighe-

den af dem er stor nok. At disse krav først bliver formuleret så relativt sent i forløbet er beklageligt for os som udviklere. Er der noget vi kunne have gjort for at undgå dette? Måske kunne vi have gået endnu hårdere til EIK Bank rådgiverne under sessionen hvor vi forsøger at fastlægge og kvantificere kravene til applikationen, hermed dog ikke sagt at vi ikke var direkte i vores spørgemåde. Denne session med prototype 2 varede i nærheden af 2 timer og der kom virkelig meget information på bordet. Det hører iøvrigt ikke til sjældenhederne med længerevarende møder. Problemet her består i at allerede efter 30 min er koncentrationsevnen dalet betydeligt, og efter $1\frac{1}{2}$ time er man nået et punkt hvor der for alvor begynder at opstå misforståelser i kommunikationen.

Kapitel 4

Rød tråd

4.1 Gennemgang af process

Dette afsnit handler om hvordan vi greb projektet an.

- **Første uge:** Vi blev installeret på kontoret og vist rundt i huset, sådan at alle i huset vidste hvem vi var. Dette var med til at få os til at føle os velkomne.

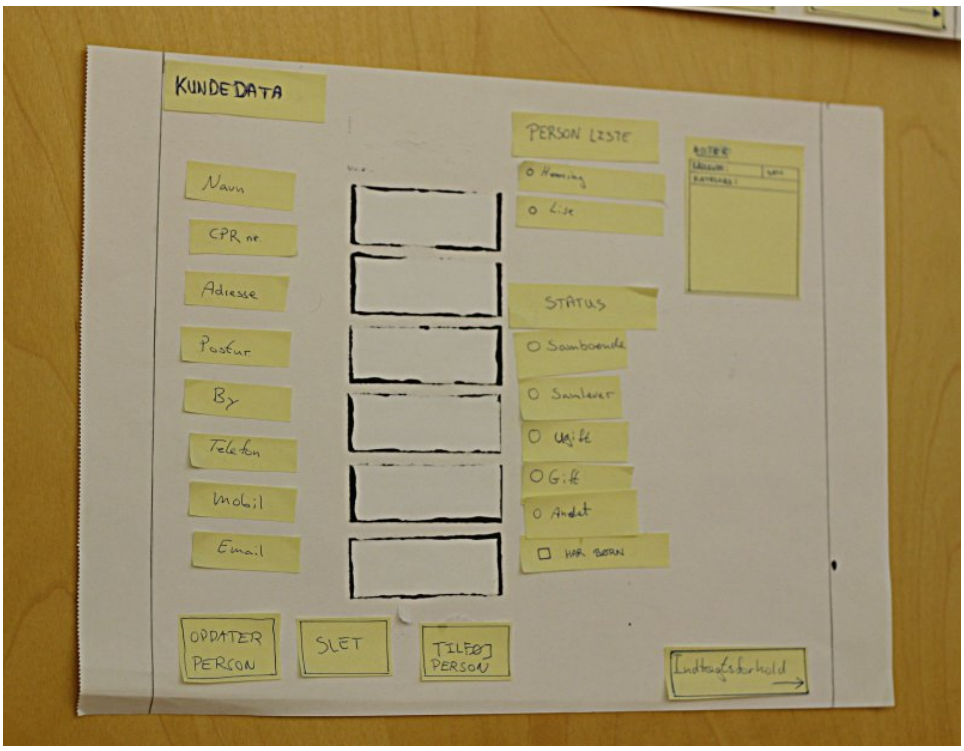
  Vi holdt et møde hvor vi som udviklere og EIK Bank hver især udtrykte ønsker og forventninger om projektforløbet. Det var her at grunden blev lagt til et ugentligt møde, hvor vi kunne fremlægge hvad vi havde lavet og hvad vi ville lave i den kommende uge.

  Den første nød der skulle knækkes omhandlede kravet om at systemet skulle bygges op omkring en central database. Grunden til at dette er den første nød er at databasen vil komme til at danne grundlag for systemet. Problemet bestod i at undersøge hvilken database vi havde til vores rådighed, og hvordan tilgangen til den ville blive. Som man kan læse i afsnit 5.2 på side 45, så havde vi flere muligheder og valget faldt på en Access database som, i form af de fremtidige udvidelsesmuligheder, vil blive placeret på EIK Bank's fil server.

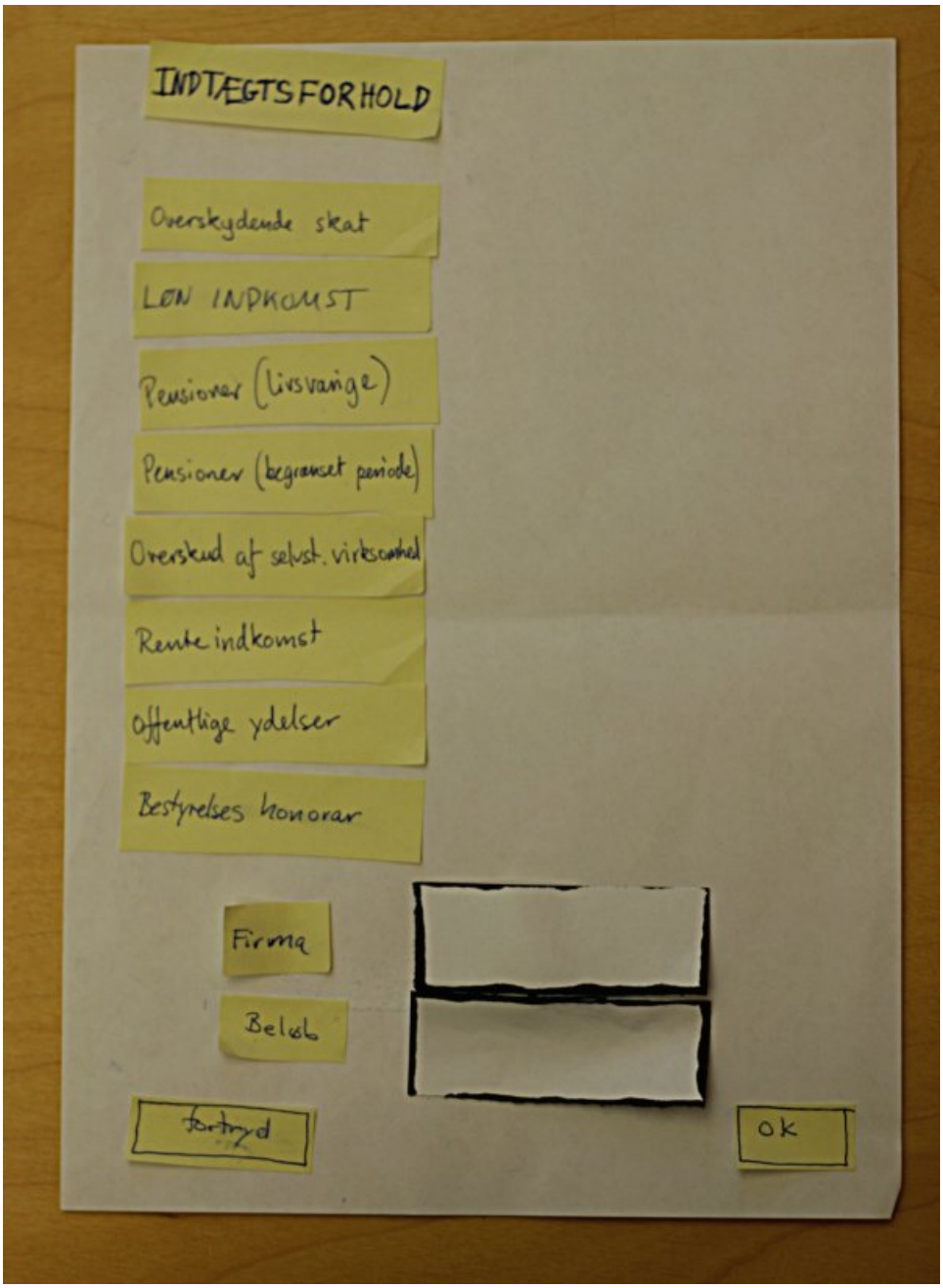
  Det var også i forbindelse med denne nød at det gik op for os, hvor vigtigt det var at definere og præcisere vores nødder og mål i en sådan grad at vi, som udviklere, ikke var i tvivl om konsekvensen af ikke at nå et mål.

  Derudover havde vi en session hvor vi kiggede en medarbejder over skulderen, for at finde ud af hvad deres nuværende arbejdsgang består af. Denne session fik vi lov til at videofilme og sammen med observation på noteform dannede det grundlag for dokumentationen af sessionen.

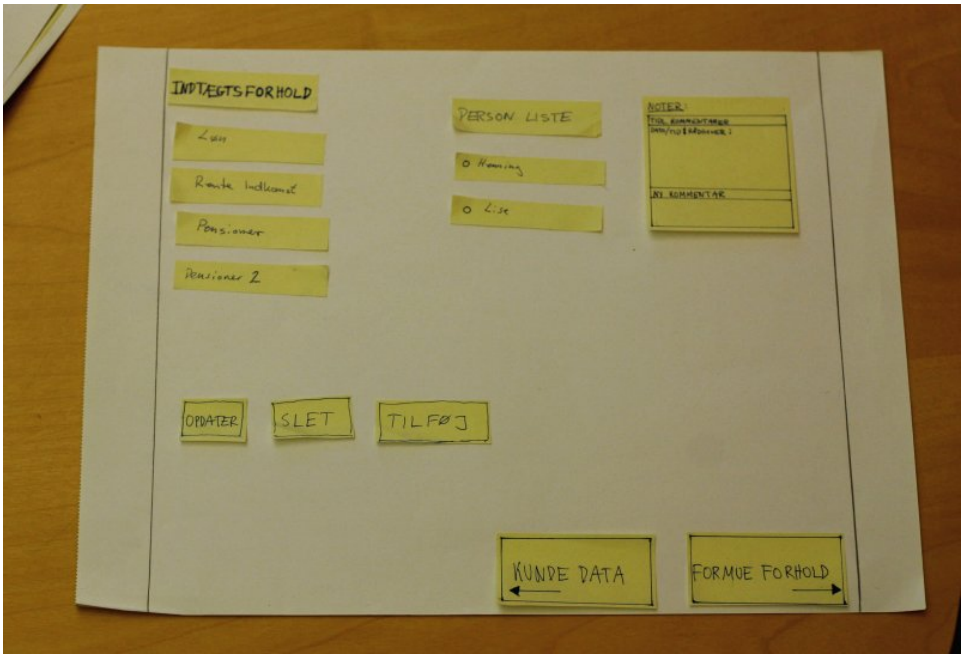
  Det var endvidere i denne uge at vi fik konstrueret den første Lo-Fi prototype.



Figur 4.1: Prototype 1, KundeData



Figur 4.2: Prototype 1, IntaegtForhold popup



Figur 4.3: Prototype 1, IntaegtForhold

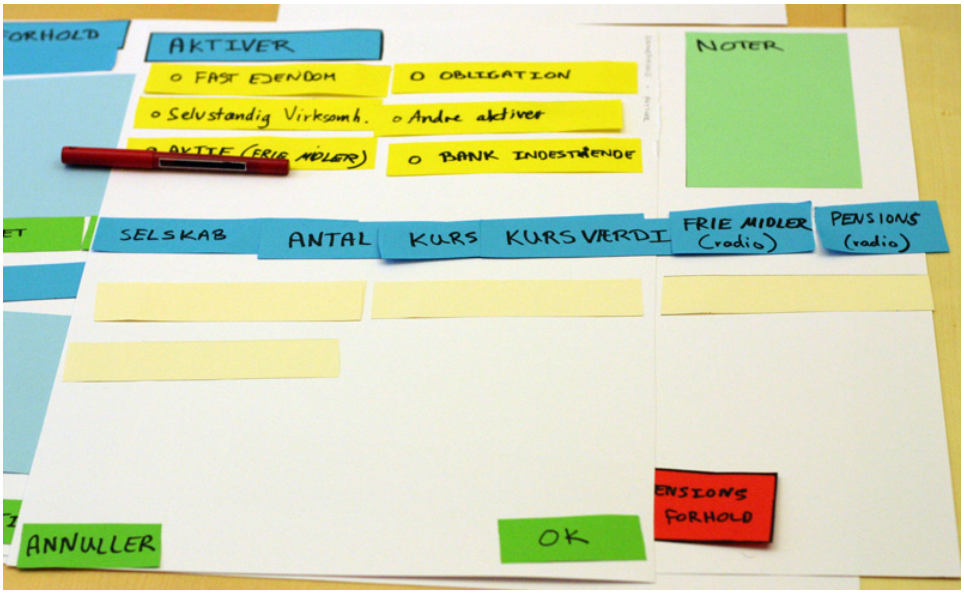
- **Anden uge:** Vi afholdte to sessioner med den første prototype, hvor målet var at få indsamlet krav til systemet, se afsnit 3.1 på side 26. Sessionerne blev dokumenteret ved hjælp af video og notetagning. Ugen blev også brugt til at analysere de selvsamme krav, og så fik vi et crash kursus i pensionsteori. Denne uges hårde nød var at indsamle krav og at få en forståelse af EIK Banks verden. Derudover blev vi einge om at fredag fremover skulle være en dag hvor vi skrev rapport.

- **Tredje uge:** Vi konstruerede anden prototype og afholdte tredje prototype session. Foto fra sessionen kan ses på side 32, 32 og 33 Til denne session valgte vi at gøre brug af lydoptagelser og noter. Det var også i denne uge at vi fandt ud af hvad vi skulle fokusere på i vores rapport. Det vil sige at vi valge to studieområder, brugevenlige brugergrænseflader og design patterns.

- **Fjerde uge:** Denne uge omhandlede primært spørgsmål om design af det nye system. Vi identificerede klasser og lavede den overordnede arkitektur til systemet. Det udmyndigede sig i et klassediagram, hvilket kan ses i afsnit 10 på side 102. Vi fik lavet en testapplikation som gjorde brug af design patternet Model View Controller. Dette kan der læses mere om i afsnit 6.3 på side 76. Denne test var grundlaget for arkitekturen i systemet. Denne uges hårde nød gik ud på at få kodet en lille applikationen som kunne læse og skrive til en Access

Figur 4.4: Prototype 2, KundeData

Figur 4.5: Prototype 2, FormueForhold popup



Figur 4.6: Prototype 2, FormueForhold popup 2

database. Denne applikation skulle derfor testes på en af EIK Banks computere og alt gik godt. Testen kan læses i afsnit 7.3 på side 86. Det var også i denne uge at lavede et strukturdokument der fortæller om navnekonventioner i koden.

- **Femte uge:** Ugen gik med med at programmere sidste uges fastlagte arkitektur. Vi diskuterede hvordan og hvorledes med performance og database. Diskutionen kan læses i afsnit 5.2.2 på side 47. Der var i denne uge ikke nogen nød, ud over det faktum, at der var kilometer vis af kode der skulle kodes og der naturligvis ikke var tid nok i døgnet.

- **Sjette uge:** Vi programmerede forsat på livet løs, men til forskel fra sidste uge fokuserede vi mere på at få koden til at virke sammen med de brugergrænseflader vi havde lavet. Det var også i denne uge at vi blev enige om at begynde at kode brugergrænsefladen til 'indtægter' og 'kunde' helt igennem, for så at kunne bruge den som skabelon for de resterende brugergrænseflader. På den måde kunne vi nøjes med at lave fejl ét sted og dermed få vigtige kodemæssige erfaringer, som så i fremtiden kunne bruges på de andre skærme. Den hårde nød gik igen fra sidste uge.

  På den ugentlige fredagsbriefing præsenterede vi EIK Bank for en afgrænsning, som gik ud på at udlade printfunktionaliteten i systemet. Det var EIK Bank ikke enige i og dermed blev vores plan ændret og vi fandt lige pludselig en ny hård nød.

- **Syvende uge:** Som reaktion på sidste uges fredagsmøde arrangerede vi et møde med EIK Bank hvor vi kvantificerede en del af kravene. Vi kom også igang med den hårdeste nød, det vil sige udprintning, fik undersøgt emnet på nettet og konsulteret en lærer på skolen. Vi fik til dels implementeret regler for brugergrænsefladen, på en sådan måde at brugeren ikke kan komme til at lave ulykker.

- **Ottene uge:** Vi arbejdede hårdt på at få knækket den hårdeste nød, udprintning. Se afsnit 5.3 på side 59. Den formatering, som vi var blevet enige med EIK Bank om, voldte os store problemer og vi blev internt i gruppen enige om at lave én printfunktion som virkede, frem for at lave noget halvfærdigt som ikke virkede. Det betyder at vi kun til dels har fået knækket nødden. Vi kan printe, men ikke helt som vi gerne skulle. Vi fik også finpudset de sidste ting i koden, lavet skærmene 'KundeData' og 'Indtaegt' færdige og vi fik skrevet kommentarer de steder hvor koden var færdig. Det var også i denne uge af vi lukkede for videreudvikling på systemet, for herefter at fokusere på rapport skrivning.

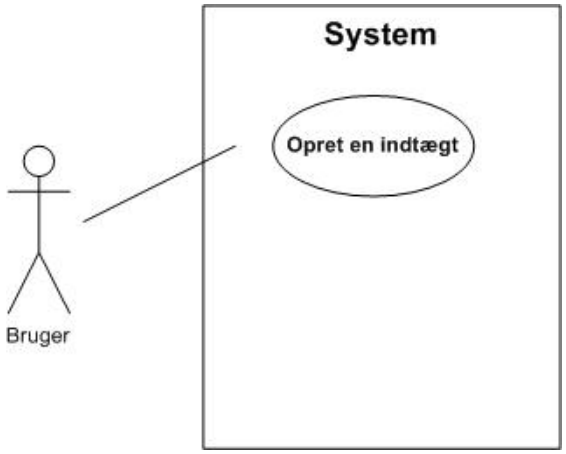
- **Niende uge:** Den hårdeste nød nu at skrive rapport og få den færdig. Vi fik også lavet en test session hvor vi ville se, om der var potentiale i vores system, angående kravet om hvor lang tid det må tage at indtaste kunde oplysninger. Vi fik afleveret en foreløbelig rapport til EIK Bank så de kunne læse den igennem og se at vi ikke har skrevet noget inkriminerende om EIK Bank.

- **Tiende uge:** Der var stadig en del der skulle skrives til rapporten og det blev gjort. Diagrammer, skærmbilleder og så videre blev gjort færdige og klar til at komme i rapporten. Nu da rapporten er færdig er den sidste hårde nød knækket.

4.2 Eksempel på dokumentation

EIK Bank har udtrykt ønske om at vi, i udviklingsperioden, fokuserer på funktionalitet i applikationen, fremfor dokumentation af systemet. For at give et eksempel på dokumentation af applikationens funktionalitet, har vi valgt at gennemgå hvordan vi opretter en indtægt, fra start til slut. Vi vil her vise relevante diagrammer, fra krav og usecase-, til klasse- og sekvensdiagrammer og demonstrere funktionens kald ned gennem arkitekturens lag. Dermed giver vi også et indblik i hvordan vores arkitektur hænger sammen.

Dokumentation er ikke helt uvæsentlig når der udvikles software til en bank. Når en bank vil tage et edbsystem i brug skal det godkendes af Finanstilsynet. Det betyder at banken skal dokumentere systemet efter gældende lovgivning.



Figur 4.7: Usecase Indtaegt

Ud fra de fastlagte funktionelle krav om hvad applikationen skal kunne, ses det blandt andet at programmet skal kunne tilføje, se, slette og redigere en indtægt i et kundeforhold. Udførelsen af dette krav sker i forbindelse med indtastningen af kundens forhold. Kravet er fundet ved at se på hvilke kundeforhold der findes i EIK Banks nuværende system. Vi har til dette eksempel valgt at bruge delkravet 'tilføj indtægtsforhold', og som indtægtsforhold bruger vi aktieindkomst. Kravet står som nummer 20 på kravlisten og ud fra det kan vi definere følgende usecase, se figur 4.7 på side 35.

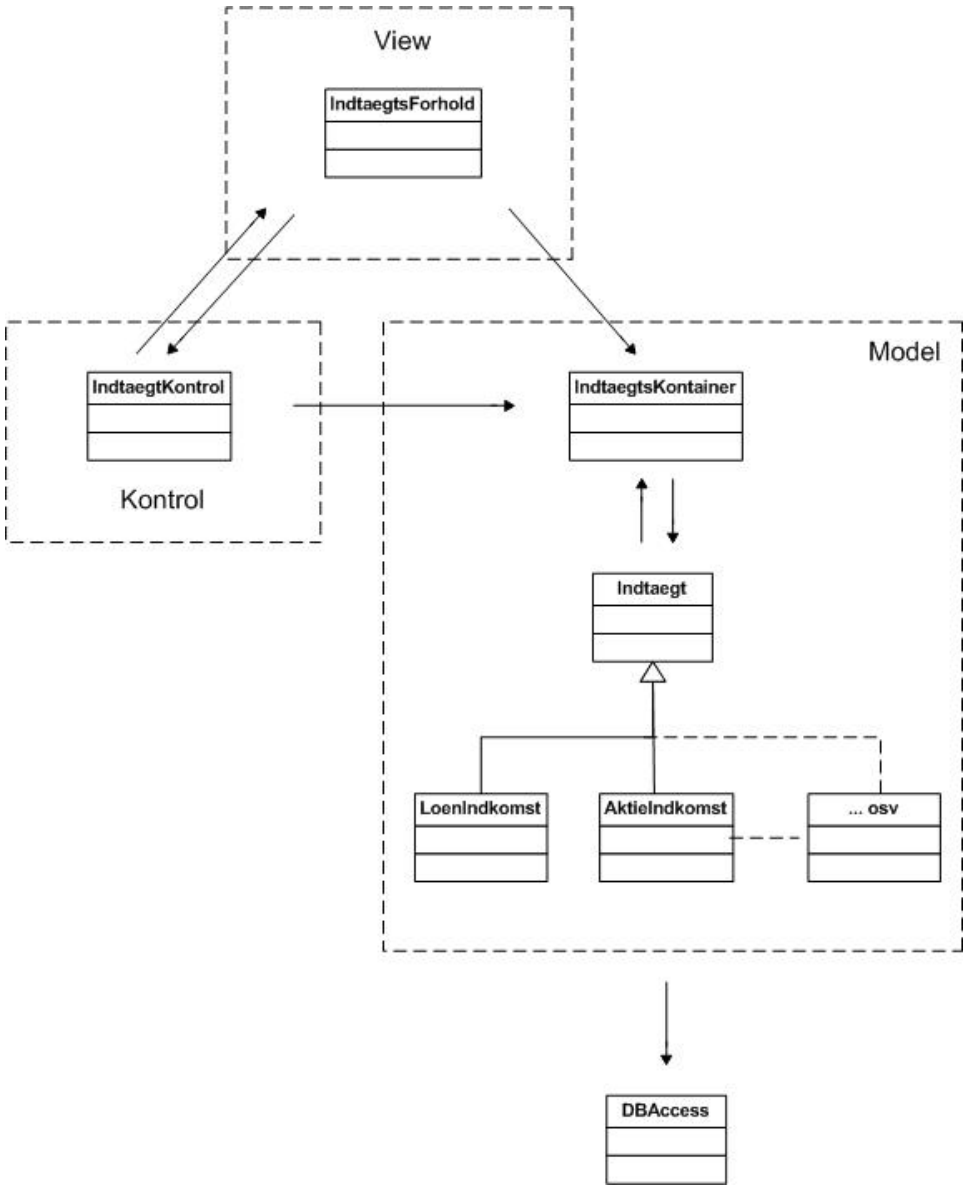
Funktionen ligger i klassen 'Indtaegt' og er navngivet *GemIndtaegt()*. Klassen 'Indtaegt' er abstrakt, hvilket betyder at man ikke kan instantiere klassen, vi instantierer subklasserne og ikke superklassen. Metoden *GemIndtaegt()* er virtuel, hvilket betyder at de klasser der arver metoden, nemlig subklasserne, har mulighed for at overskrive funktionen. Se udsnittet af klassediagrammet, figur 10.2, side 103.

Nedenstående arkitektur-diagram viser strukturen i applikationen, i forbindelse med 'Indtaegt', se figur 4.8 på side 37. Der gives derudover et eksempel fra appliktionenen på hvilken indflydelse design patternet MVC har på vores arkitektur og den opdeling det i praksis medfører. Der kan læses om MVC i afsnit 6.3.1 på side 76.

Følgende sekvensdiagram viser de specifikke kald i strukturen i forbindelse med at oprette en 'indtaegt', i dette tilfælde en 'aktieindkomst', se figur 4.9 på side 39. Af praktiske årsager, hvilket vil sige pladsmangel på siden, har vi valgt at springe visse kald over i diagrammet. Hele forløbet beskrives i nedenstående gennemgang, hvor 2 til og med 5 er udeladt fra sekvensdiagrammet:

1. Brugeren opretter en ny indtægt på formen 'Indtaegter' (view) i applikationen. Dette gøres ved at trykke på knappen 'Tilføj'.
2. Formen 'Indtaegter' sender en event videre til kontrol klassen 'IndtaegtKontrol'.
3. 'Indtaegtkontrol' kalder, via eventhandleren, funktionen *session.getKunde()* for at opklare hvilken kunde der er valgt på personlisten.
4. *session.GetKunde()* kalder *KundeKontainer.GetKunde()*, som returnerer kunde-objektet til 'IndtaegtsKontrol'.
5. Eventhandleren i 'IndtaegtKontrol' kalder nu *Kunde.GemAktieIndkomst()* med udbytte og kursgevinst som parametre.
6. *Kunde.GemAktieIndkomst()* kalder *IndtaegtsKontanier.GemAktieIndkomst()*.
7. *IndtaegtsKontainer.GemAktieIndkomst()* opretter nu en ny instans af 'AktieIndkomst', som får overført parametrene til dens konstruktør.
8. Derefter kaldes *GemIndtaegt()* på objektet 'AktieIndkomst'.
9. *AktieIndkomst.GemIndtaegt()* starter med at kalde baseklassens *GemIndtaegt()*.
10. *Indtaegt.GemIndtaegt()* sørger for at lave plads i tabellen 'Indtaegter' i databasen.
11. Derefter opdaterer *AktieIndkomst.GemIndtaegt()* den oprettede tupel i databasen, med de data der skal gemmes for aktieindkomsten,
12. Til slut opretter *IndtaegtsKontainer.GemAktieIndkomst()* objektet i datastrukturen, ved at tilføje aktieindkomsten til den generiske liste af indtaegter.

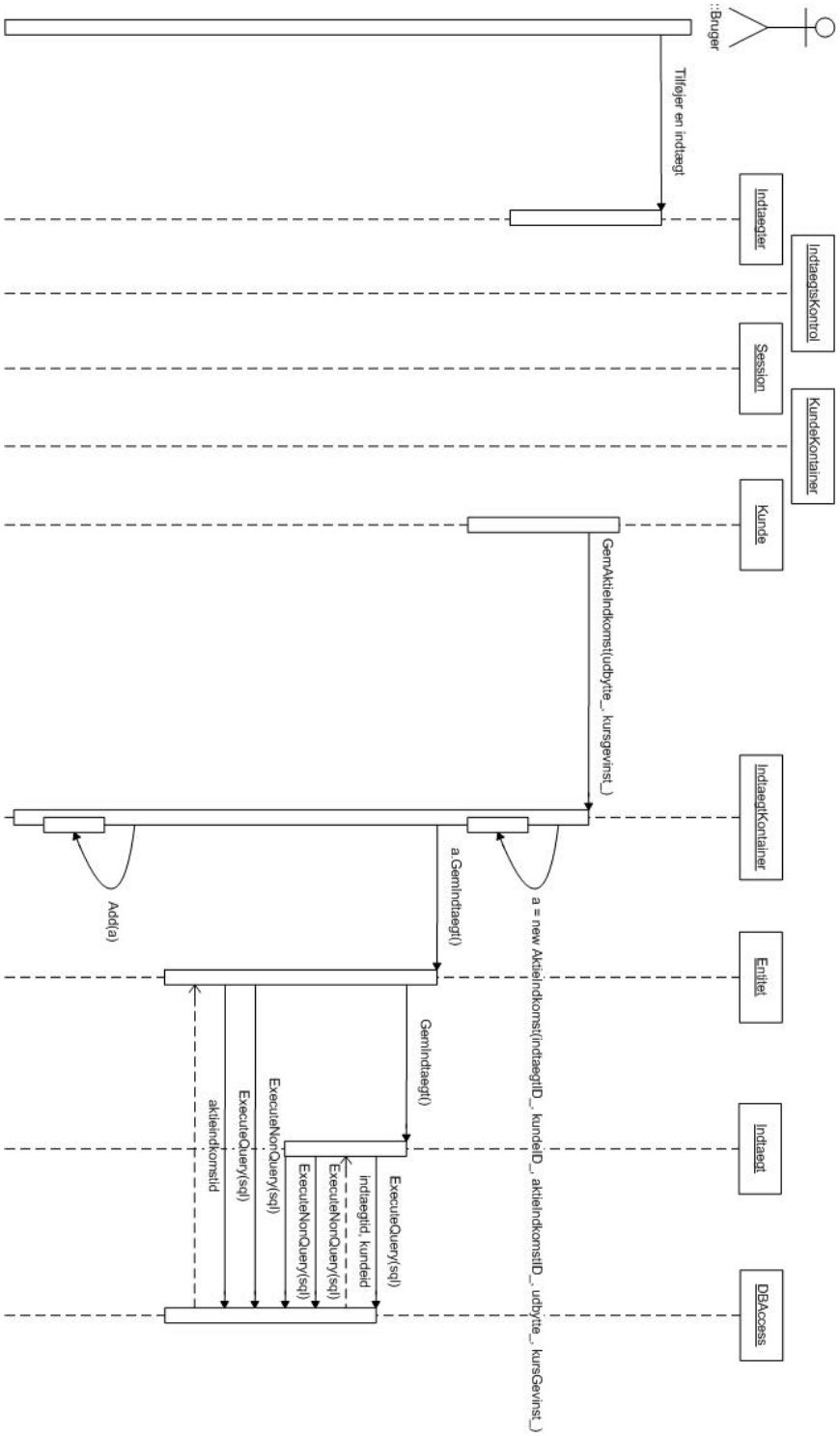
Ovenstående afspejler den algoritme vi har konstrueret til at gemme et kundeforhold. Algoritmen kan ses i 6 trin, på side 51 under afsnittet 5.2.3.



Figur 4.8: System arkitektur

Her viser vi et udsnit af koden som understøtter sekvensdiagrammet 4.9 på side 39.

```
public partial class Indtaegter : Form
{
  private void btnIndtaegterTilfoej_Click(object sender,
      EventArgs e)
  {
    indtaegtKontrol.getEvent(sender);
    set_false();
  }
}

//Følgende er et resultat af at en bruger har trykket på
    knappen tilføj på popup formen til 'Indtaegter'

public class IndtaegtKontrol
{
  private System.Windows.Forms.Button SenderBtn = null;

  private Indtaegtsforhold indtaegtForm;
  private Indtaegter indtaegtPopUpForm;
  private Session session;
  private Indtaegt.Indtaegt indtaegt;

  public void getEvent(object sender)
  {
    if (sender.GetType().Name == "Button")
    {
      //Her typecastes objektet 'sender' til typen 'Button'
      SenderBtn = (System.Windows.Forms.Button)sender;

      if (sender.GetType().Name == "Button")
      {
        SenderBtn = (System.Windows.Forms.Button)sender;
        if (SenderBtn.Name == "btnIndtaegterTilfoej" &&
            SenderBtn.Text == "Tilføj")
          {
          Kunde kunde = null;

          if (indtaegtForm.rdoPerson1.Checked)
          {
            kunde = session.GetKunde(0);
          }
          else if (indtaegtForm.rdoPerson2.Checked)
          {
            kunde = session.GetKunde(1);
          }
          indtaegtPopUpForm.Refresh();
```



Figur 4.9: Sekvensdiagram GemIndtaegt

```
            //Her springer vi lidt i funktionen for kun at
                vise det, for eksemplet, relevante kode. Det
                vil sige uvedkommende forgreninger er undladt.

            else if (indtaegtPopUpForm.rdoAktieIndkomst.
                Checked)
            {
              try
              {
                int udbytte = Convert.ToInt32(
                    indtaegtPopUpForm.txtUdbytte.Text);
                int kursgevinst = Convert.ToInt32(
                    indtaegtPopUpForm.txtKursGevinstTab.Text);
                kunde.GemAktieIndkomst(udbytte, kursgevinst);
              }
              catch (FormatException fe)
              {
                Console.WriteLine("IndtaegtKontrol.btnTilfoej
                    ␣:␣" + fe.Message);
              }
            }
            indtaegtForm.OpdaterTrae(kunde);
          }
        }
      }
    }
}

//Følgende er et resultat af IndtaegtKontrol.kunde.
    GemAktieIndkomst(udbytte, kursgevinst);

public class Kunde
{
  private IndtaegtsKontainer indtaegter;

  public void GemAktieIndkomst(int udbytte_, int
      kursgevinst_)
  {
    indtaegter.GemAktieIndkomst(udbytte_, kursgevinst_);
  }
}

//Følgende er et resultat af Kunde.indtaegter.
    GemAktieIndkomst(udbytte_, kursgevinst_);

public class IndtaegtsKontainer
{
  private List<Indtaegt> indtaegter;
```

```
  public void GemAktieIndkomst(int udbytte_, int
      kursgevinst_)
  {
    AktieIndkomst a = new AktieIndkomst(0, kundeID, 0,
        udbytte_, kursgevinst_);
    a.GemIndtaegt();
    indtaegter.Add(a);
  }
}

//Følgende er et resultat af kaldet IndtaegtsKontainer.a.
    GemIndtaegt();

public class AktieIndkomst : Indtaegt
{
  private int aktieIndkomstID;
  private int udbytte;
  private int kursGevinst;

  public override void GemIndtaegt()
  {
    ///Tilføjer en entry i tabellen Indtaegt
    base.GemIndtaegt();
    ///Tilføjer aktieindkomsten til tabellen aktieindkomst,
        med relation til
    ///den indtaegt der blev tilføjet ovenfor
    string sql = "INSERT INTO aktieindkomst (indtaegtid, "
        udbytte, kursgevinst) values (" + base.IndtaegtID +
        "," + udbytte + "," + kursGevinst + ")";
    DBAccess.getDBInstance().ExecuteNonQuery(sql);

    ///Henter det pågældende indtaegtid
    sql = "SELECT aktieindkomstid FROM aktieindkomst WHERE "
        indtaegtid=" + base.IndtaegtID;
    DataSet myDataSet = DBAccess.getDBInstance().
        ExecuteQuery(sql);

    try
    {
      aktieIndkomstID = Convert.ToInt32(myDataSet.Tables["
          Result"].Rows[0][0].ToString());
    }
    catch (NullReferenceException nre)
    {
      Console.WriteLine("AktieIndkomst.GemIndtaegt 2 : " +
          nre.Message);
    }
  }
}
```

```
//Følgende er et resultat af kaldet AktieIndkomst.base.
    GemIndtaegt();

public abstract class Indtaegt
{
  private int indtaegtID;
  private int kundeID;

  public virtual void GemIndtaegt()
  {
    ///Finder den tupel i tabellen indtaegt der er klar til
        brug (den med det højeste id)
    string sql = "SELECT indtaegtid, kunde.kundeid FROM
        indtaegt inner join kunde on indtaegt.kundeid =
        kunde.kundeid WHERE navn='Default'";
    DataSet myDataSet = DBAccess.getDBInstance().
        ExecuteQuery(sql);
    int emptyIndtaegtID = 0;
    int defaultKundeID = 0;
    try
    {
      emptyIndtaegtID = Convert.ToInt32(myDataSet.Tables["
          Result"].Rows[0][0].ToString());
      defaultKundeID = Convert.ToInt32(myDataSet.Tables["
          Result"].Rows[0][1].ToString());
    }
    catch (NullReferenceException nre)
    {
      Console.WriteLine("Indtaegt.GemIndtaegt 1 : " + nre.
          Message);
    }

    ///Indsætter en ny tom tupel i indtaegt (id+1).
    sql = "INSERT INTO indtaegt (kundeid) VALUES (" +
        defaultKundeID + ")";
    DBAccess.getDBInstance().ExecuteNonQuery(sql);

    ///Opdaterer den (tomme) hentede tupel med det
        tilhørende kundeID.
    sql = "UPDATE indtaegt SET kundeid=" + KundeID + "
        WHERE indtaegtid=" + emptyIndtaegtID;
    DBAccess.getDBInstance().ExecuteNonQuery(sql);
    this.IndtaegtID = emptyIndtaegtID;
  }
}

//Følgende er et resultat af kaldet IndtaegtKontrol.
    indtaegtForm.OpdaterTrae(kunde);
```

```
public partial class Indtaegtsforhold : Form
{
  private IndtaegtKontrol indtaegtKontrol;
  private Session session;

  public void OpdaterTrae(Kunde k)
  {
    List<Indtaegt.Indtaegt> indtaegtListe = k.
        getIndtaegtKontainer().GetIndtageter();

    treeIndtaegter.Nodes.Clear();

    foreach (EIKBank.Indtaegt.Indtaegt i in indtaegtListe)
    {
      if (treeIndtaegter.Nodes.Count == 0)
      {
        TreeNode gren = new TreeNode(k.getIndtaegtKontainer
            ().GetSumAfIndtaegt(i));
        treeIndtaegter.Nodes.Add(gren);
        TreeNode blad = new TreeNode(i.Praesenter());
        gren.Nodes.Add(blad);
      }
      else
      {
        bool found = false;
        foreach (TreeNode n in treeIndtaegter.Nodes)
        {
          if (n.Text.Equals(k.getIndtaegtKontainer().
              GetSumAfIndtaegt(i)))
          {
            found = true;
            TreeNode blad = new TreeNode(i.Praesenter());
            n.Nodes.Add(blad);
          break;
          }
        }
        if (found == false)
        {
          TreeNode gren = new TreeNode(k.
              getIndtaegtKontainer().GetSumAfIndtaegt(i));
          treeIndtaegter.Nodes.Add(gren);
          TreeNode blad1 = new TreeNode(i.Praesenter());
          gren.Nodes.Add(blad1);
        }
      }
    }
  }
}
```

Kapitel 5

Tekniske løsninger

5.1 Vores valg af udviklingsmiljø

Vi har i gruppen hver især forskellige programmeringssprog som vi brænder for og har arbejdet mest med. Nogle er til Java, nogle er til C# og andre er til noget helt tredje. Dog havde vi den fællesnævner at alle i gruppen havde det samme valgfag på 4. semester, nemlig .Net & Visual Studio 2003. Her dannede vi også gruppe, for at lave valgfagsprojekt i samme fag. I valgfagsprojektet skulle vi lave en applikation i Visual Studio 2003 og her valgte vi C# som programmeringssprog. Vi skulle endvidere bygge en Access database og fik dermed øvelse i at koble de to værktøjer sammen.

Da vi begyndte på hovedopgaven var .Net frameworket lige udkommet i version 2.0 og Visual Studio var udgivet i den nye version 2005. EIK Bank har ikke haft nogen præferencer over for hvilket sprog applikationen udvikles i, da de ikke selv har udviklere siddende i huset. Eneste anmærkning fra bankens side var at vi godt måtte tage videreudviklingsmuligheder med i overvejelserne. Derfor lod vi vores fælles erfaring fra 4. semester være et af argumenterne for at vælge .Net platformen og sproget C#. Et andet argument var at vi gerne ville arbejde mere med .Net og C# og lære platformen bedre at kende, da den er "oppe i tiden". Derfor mener vi også at vores valg er med til at "fremtidssikre" systemet, da banken engang i fremtiden med sikkerhed kan finde ekstern arbejdskraft til videreudvikling af systemet i og med at platformen er udbredt og anvendes mange steder.

5.2 Database

Vi har brugt en del tid i løbet af den første uge på af finde ud af hvilken databaseløsning vi kunne bygge EIK Banks nye system op omkring. Umiddelbart så vi tre mulige løsninger.

1. EIK Bank har i forvejen et samarbejde med SDC[1], hvor SDC leverer alt det nødvendige hardware og software. Vi tænkte at SDC måske havde eller kunne sætte en databaseserver op som applikation, som vi kunne køre op imod. Det kunne de sådan set godt, men det koster omkring 50.000,- i oprettelse og ca. 3.000,- om måneden, og det ville så være en IBM DB2 databaseserver. Det dur ikke, dels fordi at det er for dyrt (vi havde slet ikke tænkt på at det kostede penge) og dels fordi at DB2 er alt for stort i forhold til det vi skal bruge.

2. En anden løsning kunne være at sætte en lokal database server op i EIK Bank. Det har der ikke umidelbart været stemning for. Det handler mest om at der ikke er ressourcer, i form af arbejdskraft, til at vedligeholde en sådan server.

3. EIK Bank har hos SDC en filserver, som alle medarbejdere har adagang til. På denne filserver kunne vi lægge en databasefil, i form af en Access database.

Konklusionen er at vi vælger mulighed nr. 3. Dette er ikke det mest optimale valg, da der er flere problemer med Access.

For det første er der en performancemæssig faktor, hvor Access ikke leverer nogen særlig god ydelse.

Yderligere er der problemet med at en Access database, eftersom det bare er en fil, ikke tager hånd om problemet med samtidig skrivning til samme data i databasen. Der er ikke nogen applikation der tager hånd om, styrer og beskytter databasen.

Vi er stødt på flere, mere generelle, databaseproblematikker. Under tilgang til databasen kan der opstå problemer. Problemet kaldes 'Lost Update': Konsulent 1 tager data om kunde X med ud til et møde med kunde X og laver en masse rettelser. Konsulent 2 tager, i dette tidsrum, adgang til databasen og laver nogle små rettelser i selvsamme data til kunde X og tilføjer sine rettelser til den centrale database. Konsulent 1 kommer tilbage og vil nu uploade sine rettelser om kunde X, fra sin bærbare computer til den centrale database. Problemet består i at de rettelser konsulent 2 har lavet, bliver overskrevet af konsulent 1. Og dermed vil en opdatering blive tabt.

Problemet er ifølge EIK Bank ikke tilstede da to konsulenter aldrig arbejder på de samme kunder. Det vil sige at der eksisterer et 1:1 forhold

[1] Sparekassernes Data Central

mellem konsulent og kunde. Vi må derfor antage at sandsynligheden for at problemet skulle opstå i praksis er meget lille. Vi vil dog, som udviklere, mene at det er nødvendigt at tage hånd om problemet, da man ikke med hundrede procent sikkerhed kan afvise at problemet kan forekomme. Det er et fremtidsaspekt at den administrative afdeling skal stå for indtastningen, hvorfor man ikke kan afvise at en rådgiver og en administrativ medarbejder kan komme til at tilgå samme kunde samtidigt.

Når konsulenten er ude hos kunden kan konsulenten ikke vide om der vil være netadgang eller ej. Det betyder at konsulenten skal have data om kunden med hjemmefra. Når konsulenten kommer hjem igen skal han lægge de nye data ind i databasen. Vi kunne altså lave en løsning, som gemmer data lokalt på computeren, når der ikke er kontakt med den centrale database. Den gemte data skal tilføjes til den centrale database når rådgiveren kommer tilbage til EIK Bank. Senere har vi tænkt at rådgiveren kan koble op til den centrale database med et mobilmodem. Denne løsning kræver dog en del mere arbejde, i form af en webservice, der vil kræve en webserver.

5.2.1 Mapping af database

Med udgangspunkt i vores klassediagram skal vi have konstrueret en database. Dette gør vi ved at 'mappe' fra vores konceptuelle model, hvilket vil sige vores klassediagram, til en relationel model. Til dette formål benytter vi os af en 8 trins algoritme. Vi skal finde:

1. Regulære klasser. Vi har regulære klasser i form af Kunde, Indtægt, Pension, Aktiv og Passiv.

2. Svage klasser og identificering af én til mange (1:M) relationsforhold. Dem har vi ikke nogle af.

3. Èn til én (1:1) relationsforhold. Disse har vi ikke nogle af. Hvis vi ser på klassediagrammet ser vi (1:1) forhold mellem en entitet og en kontainer. Men da disse kontainerklasser ikke er regulære klasser, skal der ses bort fra dette.

4. Regulære (1:M) relationsforhold. En sådan relation findes mellem Session og Kunde. Da Kunde er på mange-siden, inkluderer vi Sessions primærnøglen som fremmednøgle i Kunde. På den måde kommer Kunde til at pege på den Session som kunden er en del af.

5. Mange til mange (M:N) relationsforhold. Disse har vi ikke nogle af.

6. Flerværdigede attributter. Disse har vi ikke nogle af.

7. Ternære relationsforhold. Disse har vi ikke nogle af.

8. Udvidet klassediagram. Dette går ud på at mappe 'specialisering og generalisering' til databasen. Der er 4 forskellige måder hvorpå vi kan gøre dette. Der tages udgangspunkt i den arv der er i Aktiv, som er superklassen, og de forskellige aktiver (aktier, obligationer mv.) som er subklasser.

A Vi har valgt denne model, hvor vi laver et relationsskema for hver klasse og inkluderer primærnøglen fra superklassen som fremmednøgle i subklassen. Det har den fordel at vi ikke får redundant eller unødige 'null' atributter.

B En anden version går ud på at lave et relationsskema for hver subklasse. Dette giver problemer når der skal søges, da der i så fald skal laves en join af subklasserne. Dette betyder at der skal søges i flere tabeller for at finde det man søger efter. Et andet problem er, at de data som subklasserne har til fælles vil blive repræsenteret redundant.

C En tredje løsning er et relationsskema for superklassen med attributterne fra alle subklasserne, plus een attribut til at skelne mellem subklasserne. Denne løsning bruges når der er tale om disjunkte attributter i subklasser, hvormed der menes forskellige, ikke-overlappende attributter. Problemet er her at der vil være unødigt mange null attributer

D En fjerde løsning er et relationsskema for superklassen med attributterne fra alle subklasserne, plus en attribut der agerer som flag til alle subklasserne. Denne løsning bruges når der er tale om overlappende attributter i subklasser. Der er de samme problemer som med punkt C. Vi kan med fordel gøre brug af denne model når Pensionsforhold og dens subklasser skal mappes.

5.2.2 Hvordan skal vi indlæse data fra databasen

Vi skal nu kigge på hvordan vi kan konstruere tilgangen til databasen, i henhold til det at læse data ind i datastrukturen, ved programmets opstart. Hvor meget skal der hentes fra databasen? Skal det hele hentes på en gang eller skal det deles op? Hvor ofte skal der hentes data? Disse er overordnede spørgsmål som vi vil besvare i det efterfølgende.

Følgende algoritme beskriver hvorledes data hentes fra databasen og oprettes i datastrukturen.

1. Hent alle sessioner fra databasen.

2. Opret en session i datastrukturen.

3. Hent de personer, fra databasen, der hører til den session

4. Opret en person i datastrukturen.
5. Hent alle indtægtsforhold, fra databasen der hører til den person.
6. Opret disse indtægtsforhold i datastrukturen..
7. Hent alle formueforhold/Aktiver, fra databasen der hører til den person.
8. Opret disse formuefhold/Aktiver i datastrukturen..
9. Hent alle formueforhold/Passiver, fra databasen der hører til den person.
10. Opret disse formueforhold/Passiver i datastrukturen..
11. Hent alle pensionsforhold, fra databasen der hører til den person.
12. Opret disse pensionsforhold i datastrukturen..
13. Hvis der er flere kunder, gå til punkt 4.
14. Hvis der er flere sessioner, gå til punkt 2
15. Færdig.

Da EIKBank i skrivende stund har omkring 1500 sessioner og hver session har 1,5 person med hver 14 forskellige forhold, så bliver det til 31500 kald til databasen. Dette skal kunne gøres mere effektivt, hvilket beskrives i de efterfølgende afsnit.

Test af tilgangen til databasen.

```
Public static void main(String[] args){
    Console.WriteLine(System.DateTime.Now);
    for (int i = 0; i < 10000; i++)
    {
        DBAccess.getDBInstance().ExecuteQuery("select * from
            kunde");
    }
    Console.WriteLine(System.DateTime.Now);
    Console.ReadKey();
}
```

```
start output : 20-02-2006 11:09:01
slut output  : 20-02-2006 11:16:53
```

I ovenstående kode bliver databasen tilgået 10.000 gange, samt at der er udskrevet start- og sluttidspunkt. Denne test er udført med en OLEDB forbindelse. Databasen ligger på samme maskine som koden. Det tog altså 7 minutter og 52 sekunder. Det giver ca. 22,7 forspøgelser pr sekund. En

loadtid på små 8 minutter er selvsagt ikke tilfredsstillende og vi skal finde en bedre løsning.

En anden ting man kunne gøre var kun at læse session og kunder ind fra databasen. Det giver så kun $1500 * 1,5 = 2250$ tilgange til databasen, som vil tage 2250/22,7 = 1 minut og 40 sekunder, hvilket heller ikke er helt godt nok.

En tredje ting man kunne gøre er at hente alle sessionerne ind og oprette dem. Derefter hentes alle personerne fra databasen og man kører så de $1500 * 1,5 = 2250$ gange gennem datastrukturen og opretter personerne. Det giver i alt to tilgange til databasen som tager under 1 sekund, men en køretid i datastrukturen som siger $f(O) = Ln(2250)$, hvor det til de to første metoder er $f(O) = 1$. Det er denne metode vi vil bruge. F(O) er store o notation.

Senere skal alle kundens forhold hentes fra databasen, hvilket giver 21 (en for hver tabel) kald per kunde, men der hentes kun data for de kunder der er i en session. Det vil giver max 42 kald til databasen. Det giver os 44 kald til databasen og en forsinket opstart på 2 sec. Hermed har vi nået en mere realistisk tilgang til databasen.

Der er ikke noget krav fra kundens side om at systemet skal være hurtigt til at starte op. Så den eneste grund til at bruge krudt på det her er, at det ikke er hensigtsmæssigt at vente 5 min. på at et program starter op. (Det forventes dog inddirekte at programmet har en "normal"responstid, hvilket vil sige at man højest skal vente i et par sekunder)

Med hensyn til en eventuelt fremtidig opgradering af databasen, er det et af kravene til systemet at det skal være tilpas modulært opbygget, så kun DBAccess modulet skal ændres. Dog er SQL-kaldene i flere tilfælde specifikke for Microsoft, hvorfor også SQL-kaldene skal revurderes og skrives om til normal SQL standard. Et eksempel på dette er vores brug af ordet 'Session' som et tabelnavn i databasen. Da dette er et reserveret ord i OLEDB (Jet x.x) driveren, skal navnet i Microsoft SQL-sætningen skrives som '[Session]'. Dette vil ikke virke med almindelig SQL-standard.

5.2.3 Hvordan skal vi skrive data til databasen?

En af grundene til at vi skal kigge nærmere på dette, er netop Access's problem med at den ikke tager højde for samtidighedsproblemet[2]. Derfor er vi nødsaget til selv at lave en konstruktion der beskytter os mod dette. Ud over dette skal vi igen overveje de performance- og sikkerhedsmæssige forhold.

Der er to måder at gribe sagen an på:

1. Der gemmes kun når en session afsluttes.
2. Der gemmes hver gang for eksempel en 'aktie' eller en 'pension' bliver oprettet og/eller ændret.

[2]samtidig skrivning til databasen

For at sikre konsistensen i databasen, er der nogle skridt der skal tages. Man kunne for eksempel gøre det på en sådan måde at der altid er en tom record i databasen, som altid har det højeste id.

Det man så skal gøre, når en ny post oprettes, er:

1. Fra databasen vælges den tomme tupel, ved: SELECT id FROM tablenavn WHERE kolonnenavn='tom'; Den skal bruges til at gemme den nye post i.

2. Man indsætter en ny tom tupel med et nyt max(id).

3. Man opdaterer den tupel med det tidligere max(id).

4. Hvis der er tale om at gemme hver gang en post oprettes, så skal denne også oprettet i datastrukturen.

Herunder ses et eksempel på hvordan koden ser ud til 'gem kunde'.

```
public void SaveKunde(int sessionID_, string navn_, long
    cpr_, string adresse_, int postnr_, int telefon_, int
    mobil_, string email_, int civilStatus_, bool harBoern_)
{
//Finder den tomme post
  string sql = "SELECT kundeid FROM kunde WHERE navn='Empty
      '";
  DataSet myDataSet = new DataSet();
  myDataSet = DBAccess.getDBInstance().ExecuteQuery(sql);
  int kundeID = 0;

  try
  {
    kundeID = Convert.ToInt32(myDataSet.Tables["Result"].
        Rows[0][0].ToString());
  }
  catch (NullReferenceException nre)
  {
    Console.WriteLine(nre.Message);
  }

  String getDefaultSessionIDSql = "SELECT sessionid FROM [
      session] WHERE raadgiver =    'Empty'";
  myDataSet = new DataSet();
  myDataSet = DBAccess.getDBInstance().ExecuteQuery(
      getDefaultSessionIDSql);
  int defaultSessionID = 0;

  try
  {
    defaultSessionID = Convert.ToInt32(myDataSet.Tables["
        Result"].Rows[0][0].ToString());
```

```
    }
    catch (NullReferenceException nre)
    {
      Console.WriteLine(nre.Message);
    }

    //Oprette en ny tom post
    String setTomtObjekt = "INSERT INTO kunde(navn, sessionid
        , postnr) VALUES ('Empty'," + defaultSessionID + ",
        800)";
    Console.WriteLine(DBAccess.getDBInstance().
        ExecuteNonQuery(setTomtObjekt).ToString());

    //Opdatere den tidligere tomme post
    sql = "UPDATE kunde SET sessionid="+sessionID_+",navn='"
        +navn_+"', cpr="+cpr_+", adresse='"+adresse_+"',
        postnr="+postnr_+", telefon="+telefon_+", mobil="+
        mobil_+",          email='"+email_+"', civilstatus="+
        civilStatus_+", harboern="+harBoern_+" where  kundeid=
        "+kundeID;
    Console.WriteLine(DBAccess.getDBInstance().
        ExecuteNonQuery(sql));

    //Sætter ind i datastrukturen.
    addKunde(sessionID_, kundeID, navn_, cpr_, adresse_,
        postnr_, telefon_, mobil_, email_, civilStatus_,
        harBoern_);
}
```

Denne algoritme virker kun når en kunde skal oprettes i databasen. Skal vi derimod oprette en indtægt, et aktiv eller et passiv, så skal algoritmen være anderledes.

Den ser sådan ud.:

1. Vælg DEN Indtægt som peger på kunden med navnet 'Default'.
2. Opret en ny Indtægt der peger på kunden med navnet 'Default' (der er nu 2 tomme tupler).
3. Opdater indtægten fra trin 1 så den peger på den aktuelle kunde.
4. Opret for eksempel en AktieIndkomst, som peger på Indtægten fra trin 1.
5. Vælg den AktieIndkomst som peger på Indtægten fra trin 1.
6. Opret AktieIndkomsten i datastukturen.

Herunder ses et eksempel på 'gem aktieindkomst'.

```
public override void GemIndtaegt()
{
//trin 1
  string sql = "SELECT indtaegtid, kunde.kundeid FROM
      indtaegt inner join kunde on indtaegt.kundeid = kunde.
      kundeid WHERE navn='Default'";
  DataSet myDataSet = DBAccess.getDBInstance().ExecuteQuery
      (sql);
  int indtaegtID = 0;
  int kundeID = 0;

  try
  {
    indtaegtID = Convert.ToInt32(myDataSet.Tables["Result"
        ].Rows[0][0].ToString());
    kundeID = Convert.ToInt32(myDataSet.Tables["Result"].
        Rows[0][1].ToString());
  }
  catch (NullReferenceException nre)
  {
    Console.WriteLine("AktieIndkomst.GemIndtaegt 1 : " +
        nre.Message);
  }

//trin 2
  sql = "INSERT INTO indtaegt (kundeid) VALUES (" + kundeID
      + ")";
  DBAccess.getDBInstance().ExecuteNonQuery(sql);

//trin 3
  sql = "UPDATE indtaegt SET kundeid=" + KundeID + " WHERE
      indtaegtid=" + indtaegtID;
  DBAccess.getDBInstance().ExecuteNonQuery(sql);

//trin 4
  sql = "INSERT INTO aktieindkomst (indtaegtid, udbytte,
      kursgevinst) values ("+indtaegtID+","+udbytte+","+
      kursGevinst+")";
  DBAccess.getDBInstance().ExecuteNonQuery(sql);

//trin 5
  sql = "SELECT aktieindkomstid FROM aktieindkomst WHERE
      indtaegtid="+indtaegtID;
  myDataSet = DBAccess.getDBInstance().ExecuteQuery(sql);

  try
  {
    aktieIndkomstID = Convert.ToInt32(myDataSet.Tables["
        Result"].Rows[0][0].ToString());
```

```
    }
    catch (NullReferenceException nre)
    {
      Console.WriteLine("AktieIndkomst.GemIndtaegt 2 : " +
          nre.Message);
    }

    IndtaegtID = indtaegtID;
    AktieIndkomstID = aktieIndkomstID;
}

public void GemAktieIndkomst(int udbytte_, int kursgevinst_
    )
{
    AktieIndkomst a = new AktieIndkomst(0, kundeID, 0,
        udbytte_, kursgevinst_);
    a.GemIndtaegt();
//trin 6
    indtaegter.Add(a);
}
```

Denne konstruktion sikrer at chancen for at to personer gemmer samtidigt på samme id, er blevet meget lille. Den gør også det at der altid er et tomt id at gemme på. Den gør det også muligt at oprette en entitet i datastrukturen med alle attributter. Alternativet er selv at skulle fastsætte id'er, men så skal man selv holde styr på hvem der kan oprette hvilke id'er og hvilken værdi de skal have, for ikke at komme til at oprette to ens id'er. Et andet alternativ er at indsætte sin data i databasen for så at hente den ud igen med det, fra databasen oprettede, nye id. Problemet er her at man ikke ved hvilket id entiteten har fået, da der kan være flere som gemmer samtidigt. Når man så spørger efter den med max(id) risikerer man at få fat i den forkerte entitet. Hvis ikke der gemmes når en entitet er oprettet, så får den ikke noget id og så kan det blive svært at identificere objektet i datastrukturen. For at summere op, så har vi valgt at gemme data i databasen hver gang man i programmet opretter en entitet, og samtidig at oprette denne i datastrukturen. Dette betyder at vi benytter os af de to algoritmer som er beskrevet ovenfor.

5.2.4 Normalisering af databasen

Når vi nu har fået lavet vores database, er det tid til at se på om den er konstrueret på en hensigtsmæssig måde. Med hensigtsmæssigt menes om den logiske databasestruktur er optimeret mest muligt. Det vil sige at finde betydningen af data og sammenhængen mellem disse. Vi skal finde og fjerne redundant data, og vi skal sikre entydig identifikation af data. Det er også hensigtsmæssigt at se på mulige fremtidige ændringer, og i den sammenhæng at se på om det er nemt eller svært at udvide med en alternativ database.

For at vi kan gøre disse ting skal vi undersøge betydningen af data. For eksempel kan vi se på hvad en kunde er i EIK Banks verden. Kort sagt består en kunde at persondata og fra nul til mange indtægter, formueforhold og pensioner. Kundetabellen, som kan ses nedenfor, er også et godt eksempel på en tabel hvor man typisk kan finde redundant data. Redundant data forkommer når to eller flere kunder har adresse i samme by og postnr. Det giver problemer når det utænkelige sker, at en by skifter navn eller at byen skifter postnr. Det betyder at opdateringen af disse data kan resultere i inkonsistent data. Til at hjælpe os med at optimere en logisk datastruktur gør vi brug af normalisering. Normalisering forgår i flere trin, startende fra første normalform til femte normalform og hvert trin sikrer at visse problemer ikke kan opstå. Når en database for eksempel er normaliseret efter tredie normalform, så opfylder den også første og anden normalform.

For at vores database kan siges at opfylde 1. normalform skal hver tupel have en entydig identifikation[3]. Det betyder at en delmængde af identifikationen ikke alene må kunne bruges som identifikation. Alle felter skal være atomariske, hvilket vil sige at de ikke må indeholde lister, arrays med flere. Det betyder også at der ikke må være en repeterende delmængde af felter i en tupel. Der skal være en fast tupel længde. Til hver tabel skal der være en fast tupel type. Det vil sige at en tupel i en tabel altid skal bestå af de samme felter.

De to sidste punkter kan vi ikke undgå at opfylde. Til session har vi valgt at lave et sessionid som identifikation på en tupel. Da en rådgiver sagtens kan være en del af flere sessioner samme dag med samme tomme note, kan vi ikke bruge nogle eller en kombination af disse felter som identifikation. Derfor bruger vi sessionid som identifikation og derved er første punkt opfyldt. Session indeholder ikke nogle lister, hvormed andet punkt også er opfyldt.

For at opfylde 2. normalform skal databasen være på 1. normalform, og alle ikke-identifikationsfelter skal være afhængige af hele identifikationen. Det betyder at hvis der er en identifikation som er sammensat af flere felter, så skal ikke-identifikationsfelterne være afhængige af alle identifikationsfelterne.

Da vi ikke har nogle sammensatte identifikationer opfylder vores database 2. normalform.

For at opfylde 3. normalform skal databasen være på 2 normalform, og alle ikke-identifikationsfelter må ikke være transitivt afhængige af identifikationen. Dette betyder at et felt i en tabel er afhængig af et andet felt i anden tabel, som så er afhængig at identifikationen.

Et klassisk skoleeksempel på dette er postnr og by problematikken, hvor by er afhængig af postnr, som igen er afhængig af identifikationen. Dermed er By transitivt afhængig af KundeID. Det man gør, og som vi også har gjort, er at skille by ud i en tabel for sig med identifikationen postnr. Postnr

[3]læs: nøgle

i kunde-tabellen laver vi om til en fremmedidentifikation.

Nedenfor ses et udsnit af vores tabeller.

```
CREATE DATABASE eikbank;

CREATE TABLE session(
  sessionid   AUTOINCREMENT PRIMARY KEY,
  raadgiver   TEXT,
  dato        TEXT, //Her kunne vi brugt et 'date' istedet
     for text, men det giver tit for mange problemer med
     computer installationer, programmeringssprog og
     databaser.
  note        TEXT
);

CREATE TABLE kunde(
  kundeid      AUTOINCREMENT PRIMARY KEY,
  navn         TEXT,
  cpr          LONGINT,
  adresse      INT,
  postnr       INT FOREIGN KEY REFERENCES postnr,
  telefon      INT,
  mobil        INT,
  sessionid    INT FOREIGN KEY REFERENCES session,
  civilstatus  INT,
  harboern     BOOL
);

CREATE TABLE postnr(
  postnr   AUTOINCREMENT PRIMARY KEY,
  by       TEXT          NOT NULL
);
```

Et andet sted i databasen hvor der har været problemer, er i tabellerne 'Aktie' og 'Obligation'.

```
CREATE TABLE aktiv(
  aktivid AUTOINCREMENT PRIMARY KEY
  kundeid int           FOREIGN KEY REFERENCES kunde
);

CREATE TABLE aktie(

⋮

  antal       INT,
  kurs        INT,
  kursvaerdi  INT,

⋮

);
```

Session			
SessionID	Raadgiver	Dato	Note
$x > 0$	Default		
$y > 0, y \neq x$	Empty		

Tabel 5.1: Session Tabel

```
CREATE TABLE obligation(
⋮
    antal       INT,
    kurs        INT,
    kursvaerdi  INT,
⋮
);
```

I og med at $kurs * antal = kursværdi$ kan vi fjerne *kursværdi* fra tabellerne og lade applikationen regne kursværdien ud når den pågældende form bliver genereret. Det er ikke hensigtsmæssigt at gemme en sum, som kan regnes ud fra de to andre tal, så længe at summen ikke skal bruges til noget senere. Vi har altså her fjernet redundant data, hvilket forhindrede tabellerne *aktie* og *obligation* i at opfylde 1. normalform.

Her stopper det for vores vedkommende. Man kunne gå videre med Boyes-Codd normalform og 4. og 5. normalform. Generelt kan man sige at man dekomponerer (nedbryder) sine tabeller når man normaliserer sin database. Problemet med 4. og 5. normalform er at de tager sig af problemer som kun opstår i meget komplekse databaser. Der kan være problemer ved at bruge Boyes-Codd normalform i kraft af det ikke altid er muligt at gå tilbage til et tidligere trin (3. normalform). Disse mulige problemer og begrundelser gør at vi stopper efter 3. normalform.

5.2.5 Start-opsætning af databasen

For at applikationen skal kunne virke skal der være noget data i databasen. Uden denne data vil algoritmerne til at gemme kunder, indtægter, aktiver og passiver ikke virke. I næste afsnit er 'x', 'y', 's', 't', 'o', 'p' og 'q' er naturlige tal og deres værdiger bliver automatisk sat i primærnøglen af databasen. Man skal selv huske at sætte de rigtige fremmednøgler. Tallet 'r' er lidt specielt da dette er et faktisk postnr og skal tastes ind manuelt.

Kunde			
KundeID	Navn	Postnr	SessionID
$s > 0$	Default	800	x
$t > 0, t \neq s$	Empty	800	y

Tabel 5.2: Kunde tabel

Intaegt	
IndtaegtID	KundeID
$o > 0$	s

Tabel 5.3: Indtaegt tabel

Aktiv	
AktivID	KundeID
$p > 0$	s

Tabel 5.4: Aktiv tabel

Passiv	
PassivID	KundeID
$q > 0$	s

Tabel 5.5: Passiv tabel

Postnr	
Postnr	By
800	Høje Taastrup

Tabel 5.6: Postnr tabel

5.2.6 Sikkerhed

En af de ting vi ikke har kigget så meget på er transaktionsstyring. Transaktionsstyring er ikke helt uvæsenligt når vi kigger på vores gem algoritme, se afsnit 5.2.3 på side 51. Da databasen tilgåes fem gange pr. gennemløb er der mange ting som kan gå galt. Hvis der går noget galt et sted, så fejler hele algoritmen. I sådan et tilfælde hvor der er skrevet noget, men ikke det hele, til databasen, ville det være mest praktisk at lave et rollback. Det vil sige at skrive databasen tilbage til tilstanden før man begyndte at skrive. Nedenfor ses et eksempel på hvordan dette kunne implementeres. Følgene kode er pseudokode.

```
IDbTransaction trans = null;
try{
  trans = connnection.BeginTransaction(IsolationLevel.
      Serializable);
  Gem algoritmen
  trans.Commit();
}
catch(exception e){
  trans.Rollback();
  Console.WriteLine(e.messege);
}
```

5.3 Udprintsfunktionalitet

5.3.1 Indledende overvejelser

Ud fra kravet fra EIK Bank om at systemet skal kunne printe en pæn rapport ud til kunden, har vi undersøgt hvilke muligheder vi har for at løse denne opgave. Vi var alle i gruppen uerfarne med det at konstruere printfunktionalitet, hvorfor en del research af området var nødvendig.

Vi lagde fokus på udprint-funktionaliteten i de sidste to uger af den tid vi havde til at udvikle systemet i. Det første vi gjorde var at undersøge muligheden for selv at konstruere en udprint-konstruktion, via de printlibraries der ligger i Visual Studio. Vi fandt dog hurtigt ud af at denne konstruktion ville blive temmelig kompliceret og kræve rigtig meget tid at lave. Kodeomfanget ville blive voldsomt og det at researche og finde ud af hvordan det kunne gøres, ville blive meget omfangsrigt. Vi konstruerede, til formålet, en helt simpel test-applikation, hvor vi printede en variabel fra datastrukturen ud. Dette virkede fint, men der er lang vej fra en simpel variabel til en konstruktionen af en færdig rapport.

Da vi ikke følte at dette var den rigtige løsning, begyndte vi at se på alternative muligheder. Vi undersøgte om der allerede findes værktøjer skabt til at løse problemet. Til det formål fandt vi, via hjælp fra en lærer på Niels Brock, værktøjet Crystal Reports. Crystal Reports er et komponentbaseret rapporteringsværktøj til .Net, med andre ord en udskriftsgenerator der kan lave formularer med datafelter fra databasen eller fra datastrukturen. Med dette værktøj kan man konstruere og designe en rapport ud fra de data man har i sin applikation, hvor den resulterende rapport minder meget om et pdf-dokument. Der er inkluderet udprint-funktionalitet i værkøtjet, hvormed printproblematikken er løst for vores vedkommende, da man kan printe rapporten ud, ved et simpelt tryk på en knap. Hermed kan programmet også levere en digital udgave af rapporten på skærmen, før man printer, hvilket var et ønske fra EIK Banks side.

Crystal Reports viste sig rent faktisk at være inkluderet i Visual Studio 2005, det udviklingsmiljø som vi har brugt til udviklingen af applikationen. Så til formålet, syntes Crystal Reports at være den perfekte løsning. Både i henhold til vores problem med selve funktionaliteten, men også i forhold til EIK Banks behov.

5.3.2 Værktøjet Crystal Reports

Crystal rapporten bliver genereret runtime, det vil sige når programmet kører og brugeren på et tidspunkt beder om at få rapporten lavet.

Rapporten tilføjes til programmet som en komponent, hvor man kan definere hvilken datakilde komponenten skal have. Vi er ikke interesserede i at have databasen som datakilde, da vi allerede har trukket data ind i datastrukturen fra databasen én gang. Dermed er det for os nærliggende at

lade rapporten få sine data direkte fra datastrukturen, også da det formentlig vil være hurtigere at tilgå datastrukturen end at tilgå databasen. Til dette formål oprettes et dataset, hvor de for rapporten, relevante data placeres. Rapporten får det oprettede dataset som datakilde og har dermed adgang til de data den skal bruge. Disse data omfatter kundens personlige data og økonomiske forhold.

Designet af rapporten kan foregå via den wizard som opretter rapporten, eller via et 'træk og slip' designview hvor man manuelt kan styre komponenterne i rapporten.

Designet af selve rapporten baserede vi på et tidligere udarbejdet layout, hvor kunderne i en session var stillet op ved siden af hinanden i kolonnner og hvor de økonomiske forhold var listet i rækker. Vi lavede dette udkast i en almindelig teksteditor, ud fra et udkast som EIK Bank selv fremstillede tidligt i projektforløbet. Derefter fremlagde vi det for EIK Bank til en fredagsbriefing, for at få deres mening om layoutet og for at vise dem hvad vi havde forestillet os i forbindelse med rapporten. EIK Bank udtrykte tilfredshed med layoutet og vi havde dermed et udgangspunkt til layoutet af rapporten som begge parter var enige om.

5.3.3 Vores erfaring med Crystal Reports

Vores arbejde med Crystal Reports viste sig dog desværre at skulle blive en del mere tungt og problemfyldt end vi havde regnet med. Det var et stort problem for os at få værktøjet til at gøre hvad vi gerne ville have det til at gøre. Vi manglede simpelthen erfaring med et værktøj, som ingen af os på forhånd kendte til. Vi førte mail-korrespence med en lærer fra skolen der har forstand på området og havde endda et møde med denne lærer. Denne hjælp fik os i gang, men det rakte desværre ikke til en fuld forståelse for værktøjets virke. Vi søgte desuden hjælp på online communitites, men det lykkedes ikke at få brudt problemet med at formatere rapporten løst.

Mere specifikt lå problemet blandt andet i at få crystal til at vise kunderne kolonnevis. Vi kunne ikke få værktøjet til at skrive andet end på rækkevis. Efter en del kæmpen med dette, valgte vi i stedet at forsøge os med at lave én rapport per kunde i sessionen. Dette lykkedes til dels, vi fik programmet til at generere en rapport per kunde, alt efter hvor mange kunder der er i en session. Der var stadig problemer med den rapport som kom ud som resultat af dette. Det lykkedes os ikke at få værktøjet til at vise sumbeløb for hver af kunderne. Vi kunne ikke få Crystal til at skrive summer af en tabel ud, uden også at skrive hver tupel i tabellen ud, hvilket ikke var hensigten.

Vi fik altså værktøjet til at hente relevant data ud af datasættet, men kunne ikke få denne data formateret og sorteret. Derfor endte vi ikke med det tilsigtede resultat.

Kapitel 6

Studieområde

6.1 Studieområder

I forbindelse med projektforløb og rapportskrivning har vi valgt at lægge fokus på to studieområder. Det primære studieområde omhandler design af applikationens brugergrænseflader, hvor vi har defineret følgende mål:

- At designe en brugervenlig og intuitiv brugergrænseflade.
- At hjælpe med til at give bankkunden et nemt og hurtigt overblik over dennes økonomi, på en præsentabel måde.
- At skabe et professionelt look på de data EIK Bank udleverer til sine kunder, i form af en økonomisk oversigt over kundens forhold.

Omfanget af teorien omkring design af brugergrænseflade er alt for stor til at det hele er relevant og anvendeligt i vores projekt, hvorfor vi kun vil arbejde inden for et afgrænset område med emnet, som afspejles af de mål vi har sat ovenfor.

Det andet og sekundære studieområde omhandler brugen af design patterns i systemarkitekturen. Det vi, i projektet, vil lægge vægt på er det design pattern der hedder 'Model View Control' (fremover MVC). Formålet med brugen af MVC er at gøre applikationens dele så uafhængige af hinanden som muligt, hvormed det bliver nemmere at skifte og ændre delene uafhængigt af hinanden. Eksempler på sådanne dele er brugergrænsefladerne og databasetilgang osv. Det er et mål fra vores side og et krav fra EIK Banks side at det nye system skal være udvidelsesvenligt, hvorfor det giver mening at implementere MVC i vores sammenhæng. Vi kan altså opnå dette ved at opdele systemet i moduler, hvilket MVC vil hjælpe med til. Vi benytter endvidere design patternet Singleton flere steder i applikationen. Singleton sikrer at der kun findes én samtidig instans af et objekt. Mere om dette under afsnit 6.3 på side 76.

Der er mange andre design patterns som kunne hjælpe med til at nå disse og andre mål, men vi har valgt at afgrænse os fra brugen af flere design patterns, både af tidsmæssige, men også af omfangsmæssige årsager.

6.2 Design af brugergrænseflade

"At designe en god grafisk brugergrænseflade indebærer delvist disciplin, videnskab og kreativitet", se kilde [4]. Med disciplin menes to ting. Den ene ting vi skal følge er de konventioner og regler der findes for brugergrænseflader til den pågældende platfom. I vores tilfælde er denne platform Microsoft Windows. Da vi udvikler i Visual Studio, som er et Windows udviklingsmiljø, får vi en del elementer foræret. Disse elementer er for eksempel skrifttyper og udseende på knapper, rullegardiner med videre. Hermed siger vi ikke at disse elementer ikke kan ændres, men det er ikke noget vi har fokuseret meget på. Den anden ting som disciplin dækker over er principperne for godt design af brugergrænseflader. Se afsnit 6.2.1 på side 66. Med videnskab menes der usability testing, som også er et område vi ikke brugt tid på. Usability dækker over hvor brugbart et system er, herunder hvor tilfredsstillende og produktivt systemet er. Man kan lade brugerne teste systemet og dermed finde ud af hvor lang tid en opgave tager, hvor mange fejl der bliver lavet, hvor stor grad af frustration der er i forbindelse med brugen af systemet og så videre. Se kilde [2]. Når det kommer til kreativitet har vi blandt andet kigget på valg af farver, valg af logoer og ikoner med videre.

6.2.1 Teori om brugergrænseflader

Der er forskel på funktionalitet og brugergrænseflade i et program. Funktionalitet dækker over hvad programmet faktisk gør, mens brugergrænsefladen dækker over hvordan brugerne interagerer med programmet, altså hvordan brugeren opfatter systemet og hvordan dette system arbejder.

Når man skal lave en god grafisk brugergrænseflade, er der 3 vigtige faldgrubber man skal være opmærksom på og forsøge at undgå, se kilde [4]:

1. At man benytter sig af en problemcentreret og ikke en brugercentreret designproces.
2. At man ikke forstår eventdreven programmering til fulde og ikke får udnyttet denne.
3. At man ignorerer affordance, metaforer og manipulation.

Yderligere findes der 10 designprincipper for brugergrænseflader, som vi kommer ind på efterfølgende, se kilde [4]. Da det ikke er alle 10 principper vi kan påføres vores applikation, vil vi kun liste dem vi har anvendt. Se afsnit 6.2.1 på side 66.

Den brugercentrerede design proces

Tidligere fokuserede udviklere udelukkende på om programmet virkede, og et godt program var et der brugte få ressourcer. Dette er essencen i den problemcentrede designprocess. Programmerne var endvidere rettet mod fagfolk

med en baggrund i computermiljøet. Idag er der rigelige computerressoucer til rådighed og langt de fleste brugere besidder ikke avancerede computerfærdigheder, se kilde [4]. Det samme kendetegner vores situation.

Vi vil i vores designproces ikke kun designe et program der virker eller bruger få ressourcer. Vi stiler mod at udvikle et program, man som bruger skal bruge mindst mulig grad af tilvænning til og uddannelse i, for at kunne anvende. Altså et program med en høj grad af brugervenlighed og intuitivitet.

For at opnå dette er det vigtigt at brugerne af systemet involveres gennem hele udviklingsprocessen. Vi, som udviklere, skal prøve at forstå deres arbejdsopgaver, det værktøj de bruger og den måde de tænker på i arbejdssituationerne. For at kunne opnå denne forståelse kræver det at vi har flere testsessioner med brugerne, hvor vi får mulighed for at lade brugerne få indflydelse på de justereringer vi laver på vores design. Se kilde [4].

Jo bedre vi forstår brugerne og jo mere de er involverede i udviklingen, des bedre vil vi kunne konstruere vores system.

Eventdreven programmering

Vi vil gøre brug af eventdreven programmering i projektet. Eventdreven programmering handler om at der kan ske hændelser i programmet, som udløser en effekt. En event kan betegnes som en hændelse eller en handling, og kan være en indtastning på tastaturet eller et museklik. Disse events aktiveres af brugeren af systemet. Hvis systemets events konstrueres på en hensigtsmæssige måde, kan det hjælpe med til at sikre at brugergrænsefladen ikke låser brugeren og bestemmer hvordan brugeren skal interagere med systemet. Der skal ikke være nogen predefineret menu, som bestemmer hvordan der manøvreres rundt i programmet. Programmet skal ikke give brugeren besked på at indtaste data eller bevæge sig efter et fastlagt mønster. Derimod bør brugeren kunne springe frem og tilbage i programmet. Dette sker netop ved at udføre events i brugergrænsefladen.

I det der kaldes et proceduredrevent program, løses en opgave trin efter trin og typisk efter et fastlagt mønster. Bruger indtaster data når programmet giver besked om det og brugeren kan ikke springe frem og tilbage i programmet. Manøvrering forgår gennem en predefineret menu. Både proceduredrevne og eventdrevne programmer når frem til samme resultater, forskellen ligger i hvordan brugeren interagerer med og styrer programmet.

De gode eventdrevne programmer omfatter brugercenteret design, og de bedste af disse ligger stor vægt på brugernes opfattelse og viden om det miljø de færdes i. Med viden menes der den specialistviden de har om det fagområde de arbejder med. Som udvikler vil ens programmer forbedres, hvis man forstår ens brugeres opfattelse og viden. På baggrund af dette kan man konstruere genkendelige billeder, der giver mening for brugerne.

Affordance, metaforer og manipulation

Affordances kan betegnes som egenskaber for et objekt, der indikerer hvordan man kan interagere med objektet, se kilde [4]. I den virkelige verden kan man 'interagere' med objekter eller genstande ved fysisk kontakt. I computerens verden kan et eksempel være opfindelsen af 'skraldespanden' på computerens skrivebord. For at komme af med dokumenter smides de i skraldespanden. For at slette dokumentet helt, tømmes skraldespanden. For at gendanne dokumentet ledes skraldespanden igennem. Dette er helt paralelt med den virkelige verden, hvor vi ikke ville sige til skraldespanden at den skal tømme sig selv, men derimod, ved fysisk kontakt, tage fat i den og tømme den.

En museknap har den affordance at den kan trykkes ned og dermed produceres der et museklik. Museknappen har ikke andre affordances, der kan eksempelvis ikke trykkes på en bogstavstast. Selve musen er dog et objekt med flere affordances. Selvom den måske passer fint i en menneskehånd er det ikke umiddelbart logisk at den skal bevæges over en overflade før den udfører sin funktion. En mus er nem at bruge for langt de fleste, men først efter man har lært eller opdaget dens affordances.

En metafor er en billedlig sammenligning mellem to tilsyneladende forskellige kontekster. Et eksempel på en metafor i vores sammenhæng, kan være det plus vi har anvendt som ikon på knappen 'Tilføj', se figur 6.1. Helt overordnet og uden for kontekst signalerer plus at addere. I vores system bruger vi dette plus til billedligt at signalere at man med knappen kan addere.

Med manipulation menes der interaktion mellem bruger og system. Man taler om direkte manipulation. I den virkelige verden, er dette den fysiske manipulering, med eksempelvis skraldespanden. I computerverdenen er direkte manipulation det samme, nemlig at man direkte eller "fysisk" manipulerer skraldespanden.

Gode udviklere udnytter brugernes viden om affordances fra den virkelige verdens objekter, for at lave visuelt meningsfyldte og intuitive miljøer. Dette kan opnås ved brug af metaforer. Et sådan design forbedrer brugerinterfacet betydeligt, fordi brugeren hurtigere opfatter meningen. Et sådan design er endvidere langt mere intuitivt end kommandoliniemiljøer med knapper og menuer.

Det er vigtigt at studere sine brugere grundigt og at finde metaforerer der giver mening for dem. Platformens muligheder for direkte manipulation skal udforskes, som for eksempel det at kunne udføre en "drag and drop" hændelse.

Figur 6.1: Ikoner på knapper.

Principper for godt design af brugergrænseflader

1. De interagerbare elementer der indgår i et skærmbillede kaldes for "widgets". Widgtes kan være knapper, menuer, textbokse, scrollbarer med flere. Brugeren skal uden at blive distraheret kunne forudse hvad den pågældende widget kan, ud fra dens udseende. Dette kan man kalde "Princippet om konsistens på widget-niveau", se kilde [4]. Princippet går ud på at widgets af samme art skal opføre sig ensartet. Hvis en knap reagerer på ét museklik, så skal alle knapper, af samme art, reagere på ét museklik. Hvis applikationen kræver en ny widget, som opfører sig anderledes end en typisk eller nærtbeslægtet widget, bør man give den nye widget et distinkt udseende, for at komme eventuel forvirring i forkøbet. For at få en widgets udseende til at være selvforklarende, kan man, ved hjælp af metaforer, bruge billedlig sammenligning og genkendelse.

   Som det kan ses på figur 6.1 på side 66 har vi valgt at sætte ikoner på udvalgte knapper. Målet er, ved at bruge metaforer, at fremme brugerens intuitive forståelse af hvad knappen gør. Problemet er i vores situation blot at vi ikke, igennem sessioner og samtaler med brugerne, i tilstrækkelig grad har undersøgt om de ikoner vi har valgt har samme metaforiske betydning for brugerne som for os. Dette giver en hvis grad af usikkerhed omkring vores valg af metaforer. Vi skal ikke udvikle brugergrænseflader ud fra vores egne opfattelser af metaforer, men inddrage brugeren og observere og bruge deres opfattelser. Under test i afsnittet 7.2 på side 82, kan der læses om hvordan testen af vores applikation gik.

   Vi har valgt at lave ikoner til 3 knapper, som alle går igen på de skærmbilleder der er i systemet. Knappen 'Tilføj', hvis formål er at tilføje en kunde eller et økonomisk forhold, er designet med et grønt plus-ikon. På knapper, hvis funktion er at slette en kunde eller et økonomisk forhold, har et rødt kryds som ikon.

2. Brugeren skal kunne forudse et programs opførsel, ved brug af sine erfaringer tilegnet fra andre programmer. Dette kaldes "Princippet om konsistens på platformniveau". Konsistens er ikke kun vigtig i forbindelse med widgets, men også i forbindelse med abstraktioner såsom

musebevægelser, menuplacering, ikoner og toolbarstil. Der er mange beslutninger vedrørende brugergrænseflader som er specifikke for den enkelte platform. Det er en god ide at finde og følge en god guide til design af brugergrænseflader, rettet mod den platform der arbejdes med. Hvis man føler trang til at forbedre på konventionerne, risikerer man at forstyrre brugernes vante opfattelser, hvilket netop er en af fordelene ved konventionerne; nemlig at sikre konsistens og standardisere design af brugergrænseflader. Hvis man udvikler på tværs af platforme er det vigtigt at holde konsistens med "moderplatformen", for ikke at opnå ukonsistens af applikationen på tværs af platformene. Pointen er at brugeren skifter applikation på samme platform, langt hyppigere end de bruger applikationen på tværs af forskellige platforme [4].

Som nævnt tidligere udvikler vi i et udviklingsmiljø til Microsoft Windows, nemlig Visual Studio. Det er denne platform som vores brugere er vant til at bruge og vil komme til at bruge indenfor applikationens levetid. Ved at bruge Visual Studio får vi en masse ting omkring for eksempel printere og printdialoger til vores rådighed. Dette betyder at vi med minimale anstrengelser kan lave Windowsspecifikke skærmdialoger. Hermed opnår vi den fordel at vores brugere ikke skal sætte sig ind flere nye ting end højst nødvendigt.

3. Enhver brugeradvarsel og fejldialog, som opstår på grund af applikationen, skal ses som en chance for at forbedre brugergrænsefladen. Gode brugergrænseflader har sjældent brug for programadvarsler eller fejldialoger. Undtagelsen er selvfølgelig hvor man ikke kan undlade at meddele om fejl, eksempelvis i forbindelse med hardwarefejl, diskfejl, tabt netforbindelse eller advarsler om at det skridt brugeren er ved at tage er irreversibelt og/eller kan medføre fejl. Ellers ses fejldialoger i brugergrænseflader som en designfejl. Det er bedre at forebygge, frem for at brokke sig over, at brugeren laver fejl. De hyppigste fejl kommer ofte fra ikke-korrekt formateret brugerinput i en uheldig rækkefølge. Dermed er det hensigtsmæssigt at designe brugergrænsefladen til at hjælpe brugerne med at indtaste korrekt data. Hvis programmet kræver formaterede data (datoer, møntenheder med mere) så kan man bruge 'begrænsede' widgets til input, som på hensigtsmæssig måde begrænser brugerens inputmuligheder. Hvis et bestemt programtrin ikke kan udføres ordentligt før brugeren fuldfører andre programtrin fuldstændigt, kan man så vidt muligt skjule det afhængige trin, indtil alle afhængige forhold er fuldførte. For eksempel bruger de fleste brugergrænseflademiniljøer nedtonede widgets til at signalere at den pågældene widget ikke kan vælges på nuværende tidspunkt. Man kan altså bruge nedtonede widgets til at begrænse brugerens handlinger til de tilladte.

   Her kan vi gøre to ting.

- Det skal synliggøres hvornår en knap er aktiv og hvornår den ikke er aktiv. Med andre ord, hvornår man kan trykke på en knap og hvornår man ikke kan. Dette kan der læses mere om i afsnit 6.2.3 på side 75.
- Det andet vi kan gøre er at sikre at felter som for eksempel telefon og personnummer kun kan tage imod tal. Ved at tjekke et felt hver gang noget bliver sat ind i det, kan vi sikre at der kun bliver tastet lovlige tegn. Skulle der eksempelvis blive tastet et bogstav i et felt der kun tager heltal, kan vi slettet dette bogstav. Fra brugerens synspunkt vil det se ud som om at programmet ikke reagerer på andet end tal. Dette gør at der ikke er behov for unødige popup advarsler, hvilket er med til at sikre lovlige indtastninger. Punktet er dog blevet nedprioriteret i vores system, hvorfor vi ikke har fokuseret på området.

4. Vi skal stræbe efter at få vores applikation til at være selvforklarende. Gode applikationer har udførlige manualer og online hjælpematerialer, der forklarer program features og hvordan de bruges til at løse problemer i praksis. De bedste programmer er dem hvor brugeren sjældent har brug for at benytte manualen eller slå op på nettet. Forskellen mellem god og bedst afgøres ofte af hvor selvfoklarende applikationens brugergrænseflade er. Lige fra valget af labels, skriftstørrelser og fonttyper, til måden hvorpå widgets arrangeres på skærmen. Alle designbeslutninger der træffes skal testes af brugere. Målet er at lave en brugergrænseflade der ikke kræver unødige forklaringer og dermed distraherer. Brugergrænsefladen skal være selvforklarende for en bankrådgiver, men ikke nødvendigvis være det for en avisredaktør.

   Det vi har gjort i denne situation er at afholde prototypesessioner med brugerne for at finde ud af hvilke ord de sætter på deres arbejde og de processer de er involveret i. Hermed har brugerne i vores system bestemt navngivningen af indholdet i systemet, og dermed har de langt hen af vejen selv leveret teksten til de labels vi benytter. I et par tilfælde har dette ført til at vi som udviklere er blevet lidt forvirrede omkring navngivningen. For eksempel hvornår det hedder et firma og hvornår det hedder et selskab. Det vigtigste her er dog brugbarheden af systemet og at kunden og brugerne er glade.

5. Vi skal undgå tvungen opførsel og adfærd. Programmer der er præget af tvungen opførsel, tvinger brugeren til at udføre opgaver i en bestemt rækkefølge eller på andre måder ændrer på brugerens forventede respons og adfærd. Tvungen opførsel giver brugeren en generel fornemmelse af ubehag, fordi de begrænser mere intuitive og naturlige handlinger. Brugt med omtanke kan tvungen opførsel dog have sine

fordele. For eksempel i forbindelse med 'wizards', som gennem tvungen opførsel kan simplificere og guide en bruger gennem komplekse opgaver. Advarsler og fejlmeddelser er ligeledes ofte tvungne, hvilket tvinger brugeren til at tage stilling til et kritisk problem inden de kan vende tilbage til opgaven igen. Tvangen er i dette eksempel nødvendig, men ødelægger også brugerens koncentration og målrettede opførsel, og er derfor en grund til at undgå unødvendige advarsler og fejlmeddelelser.

De bedste tvungne opførsler er afdæmpede, men ikke skjulte. Eksempel: I et typisk tegneprogram giver valget af en widget en ændret funktion, og giver derfor tvungen opførsel. Der er en tekstfelt-widget til at skrive tekst med, der er en pensel-widget til at male med, der er en figur-widget til at lave figurer med. Dette giver sjældent problemer, fordi den tvungne opførsel er bygget på en analogi fra virkelighedens verden. Gode brugergrænseflader ændrer musemarkøren for at give yderligere visuel feedback når et valg er truffet. Så hvis applikationen absolut kræver tvungen opførsel, skal man sørge for at binde den til en stærk metafor og giv brugeren visuel feedback, hvormed det kan forløbe mere naturligt og ligefremt.

Vi gør, til dels, brug af tvungen opførsel, når brugeren eksempelvis skal oprette en indtægt. Først vælger man hvilken kunde indtægten skal tilhøre, trykker tilføj, vælger indtægten, indtaster de relevante data og trykker tilføj. På den anden side har vi en navigationsbar som gør at brugeren frit kan springe mellem de forskellige skærme, som på den måde sikrer at systemet har en vis fleksibilitet når det kommer til forskellige arbejdsformer.

6. Brugergrænsefladen skal designes så brugeren kan udføre sine opgaver uden at ligge synderligt mærke til brugergrænsefladen i sig selv. Dette kunne kaldes 'Princippet om gennemsigtighed'. Grænsefladegennemsigtighed sker når brugerens opmærksomhed rettes væk fra selve grænsefladen og naturligt rettes mod opgaven. Dette opnås gennem flere faktorer, inklusivt et skærmlayout hvor både værktøj og information er hvor brugeren forventer de skal være, hvor ikoner og labels giver mening for brugeren og hvor de brugte metaforer er let genkendelige og dermed lette at lære, huske og udføre. At vælge gode metaforer og følge ovennævnte principper er en vigtig start, men den mest direkte måde at sikre en gennemsigtig brugergrænseflade er ved at udføre brugertests gennem hele programmets udviklingsforløb.

   Dette er en af de ting vi kunne have gjort anderledes. Vi har brugt tid og kræfter på at få information omkring hvad systemet skal kunne, men vi har ikke brugt tid nok på at undersøge hvor på skærmen brugeren helst ser knapper og elementer placeret. Brugerne har haft mulighed

for dette under prototypesessionerne, men der har ikke været fokuseret på det.

Da EIK Banks medarbejdere fra private banking er vant til regnskab og regenark, kunne det være en fordel hvis vi fik medbragt elementer fra regneark. Dette kunne være datagrids i stedet for tekstfelter, i forbindelse med indtastning. Generelt er der, hos afdelingens medarbejdere, en forkærlighed for at få stillet ting op i rækker og kolonner.

6.2.2 Menneskelig opfattelse

Når mennesker skal opfatte mange elementer på en gang har vi en tendens, nogle siger at det er et overlevelsesinstinkt, til at gruppere og forudindtage synsindtryk. Ser vi en del af noget, er vi meget hurtige til ubevidst at bygge videre på dette synsindtryk. Dette gør vi for hurtigt at kunne reagere på en fare.

Når det kommer til at tælle mange elementer gør vi brug af forskellige teknikker til at simplificere opgaven. En af disse teknikker går ud på at gruppere elementerne. Skal vi for eksempel tælle prikker på et stykke papir, kan vi hjælpe os selv ved at sætte ring omkring fem tætliggende elementer. Derefter kan man så tælle ringene, gange med 5 og lægge en eventuel rest til. Der er også forskellige teknikker til at gruppere elementer og disse teknikker kommer vi indvpå i afsnittet om Gestaltlovene 6.2.3 på side 70.

Et andet område hvor vi som mennesker laver opdelinger, er når husker tal. For eksempel kan det være svært at huske et telefonnummer. Dette kan gøres nemmere ved at dele telefonnummeret op i mindre dele. For eksempel 12 34 56 78 eller 123 456 78 eller 12 345 678, hvor de to sidste opdelinger ikke er så gode, da vores telefonnumre ikke er delelige med 3. Tricket består i at gruppere på en sådan måde at grupperne er mindst mulige, samtidigt med at der er så få grupper som muligt. Disse to krav er i modstrid med hinanden og det er et kompromis mellem disse der får metoden til at virke. Et andet eksempel på hvor vi deler en opgave op er regnestykker. For eksempel er $14 * 5 = 70$, men dette er måske ikke lige til at se. Deler vi derimod opgaven op på en anden måde bliver det måske lidt nemmere. For eksempel ved at sige $10 * 5 + 4 * 5 = 50 + 4 * 5 = 50 + 20 = 70$ eller ved at indse at 14 er en mindre end 15 og så sige $5 * 15 - 5 * 1 = 75 - 5 * 1 = 70$.

6.2.3 Gestaltlovene

Generel teori omkring gestalt

Gestaltlovene handler om opfattelse af helheder i et billede. Gestalt er en fri oversættelse af det tyske ord gestellt, der betyder 'opstilt'. Der er mange gestaltlove, der tilsammen beskriver hvordan man kan designe indholdet på skærmbilleder til at hænge sammen. Man kan sige at et godt design hjælper

brugeren ved ikke visuelt at distrahere denne fra den egentlige arbejdsopgave. De regler der er relevante i forhold til vores projekt beskrives som følger:

- **Loven om nærhed:** Det er vigtigt at elementer der har relation til hinanden, er placeret tæt ved hinanden og dermed opfattes som sammenhængende. Man kan altså danne en gruppering af elementer, ved at placere dem nær hinanden. For eksempel kan en 'label' (læs: tekst), der er placeret for langt fra dens tilhørende 'textbox' (læs: tekstfelt), resultere i at de to elementer ikke opfattes som sammenhængende. Man risikerer yderligere at den pågældende 'label' ligger for tæt på et andet element som den ikke har nogen relation til. Dermed kan deciderede misforståelser opstå, hvor brugeren tror at der skal tastes i et andet felt end det tiltænkte. Loven bruges hermed til at vise hvilke elementer der hører sammen. Dette kan opnås ved et samspil mellem velafstemte mellemrum og brug af rækker og kolonner.

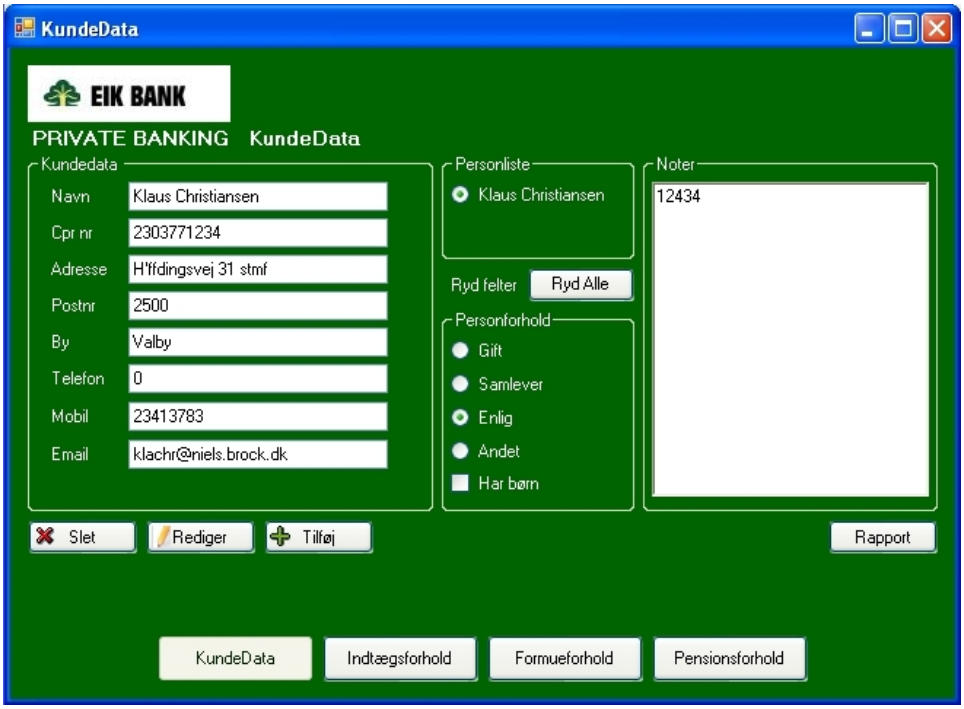
- **Loven om lukkethed:** For at tydeliggøre sammenhængen af et skærmbilledes elementer, kan man lukke grupper af elementer inde i en ramme, en såkaldt 'groupbox' (læs: ramme). En fornuftig brug af rammer, kan være med til at højne visualiseringen af sammenhængende elementer. Med en fornuftig brug menes at det ikke nytter noget at bruge for mange rammer på samme skærmbillede, hvilket resulterer i et rodet layout.

- **Loven om lighed:** Man grupperer normalt elementer efter hvor meget de ligner hinanden. Dette betyder at elementer der ligner hinanden er placeret tæt op af hinanden, for at tydeliggøre deres betydning. Rent visuelt kan dette eksempelvis opnås ved brug af fonte, font størrelser og farver. Et eksempel på dette kan være grupperingen af knapperne i værktøjslinien i Word, hvor sidestilnings-knapperne ligner hinanden og dermed grupperes sammen.

- **Loven om symmetri:** Denne regel omhandler brugen af linier og symmetri i skærmbilledet. Dette betyder at billedets elementer helst skal flugte med og være visuelt knyttet med hinanden, for at skabe et indtryk af balance og helhed. Hvis der er en tydelig asymmetri i skærmbilledet, vil dette forvirre og distrahere brugeren.

- **Loven om kontinuitet:** De indirekte linier mellem skærmens elementer, som er baseret på kontinuitet, giver en øget opfattelse af sammenhæng og gruppering. Mennesket har tendes til at følge linier, eksempelvis pile der peger i en retning.

Gestaltlovene omfatter generelle elementer i brugergrænseflade så som menuer, knapper og ikoner, grafer, tabeller, tekst. De forskellige love skal



Figur 6.2: Gestalt i vores gui.

ikke opfattes som særskilte og isolerede fra hinanden, men nærmere som sammenhængende. Sammen giver de en høj grad af logisk visualisering. Det vil sige at gestaltteorien følger princippet om at af helheden er større end summen af delene, se kilde [7].

Vores brug af Gestaltlovene

Vi har i vores system brugt de gestaltlove som var relevante for vores system. Nedenfor kan ses eksempler på hvor vi har fundet anvendelse for af forskellige love.

1. Nærhed: Navigeringsknap-menuen.
2. Lukkethed: Groupboxes.
3. Lighed: Indtastnings felter til kunde oplysninger.
4. Symmetri: Navigeringsknap-menuen.
5. Kontinuitet: Navigeringsknap-menuen.

Se figur 6.2 på side 72.

Fremtoning

Ny session
Først når der skrives i textbox ⇒ OK knap enables.

Indtægtsforhold
Når ingen person er valgt ⇒ 'Slet', 'Rediger' og 'Tilføj' gråtones og disables.
Når en person vælges ⇒ 'Tilføj' optones og enables.
Når en indtægt vælges ⇒ 'Slet' og 'Rediger' optones og enables.

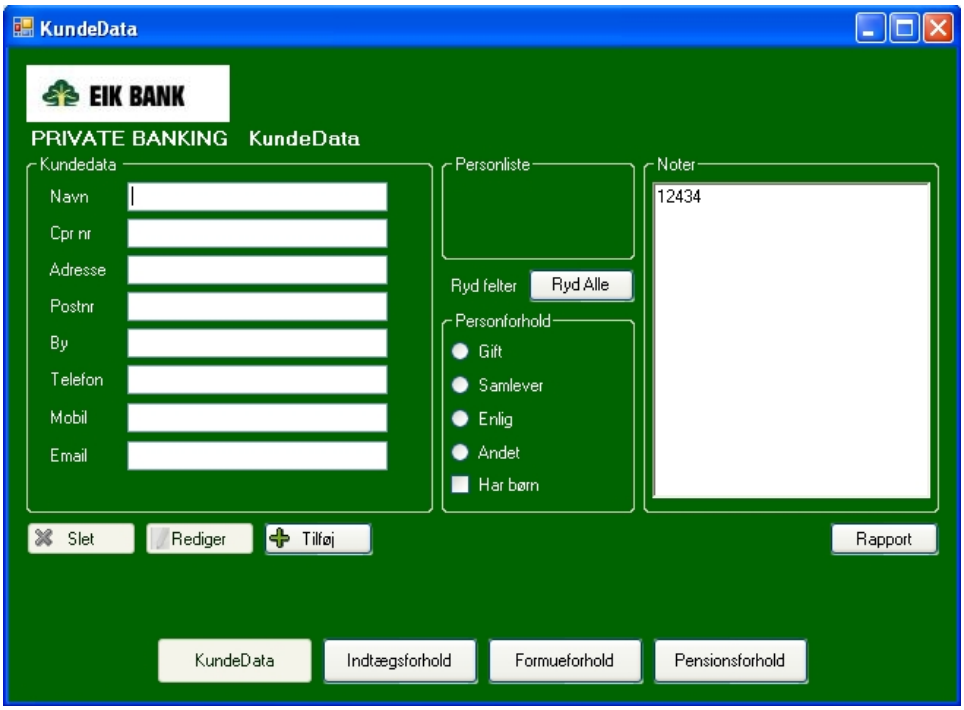
Formueforhold
Når ingen person er valgt ⇒ 'Slet', 'Rediger' og 'Tilføj' gråtones for både 'Aktiver' og 'Passiver'.
Når en person vælges ⇒ 'Tilføj' optones og enables for både 'aktiver' og 'passiver'.
Når et aktiv vælges ⇒ 'Slet' og 'Rediger' optones og enables.
Når et passiv vælges ⇒ 'Slet' og 'Rediger' optones og enables.

Pensionsforhold
Når ingen person er valgt ⇒ 'Slet', 'Rediger' og 'Tilføj' gråtones og disables.
Når en person vælges ⇒ 'Tilføj' optones og enables.
Når en pension vælges ⇒ 'Slet' og 'Rediger' optones og enables.

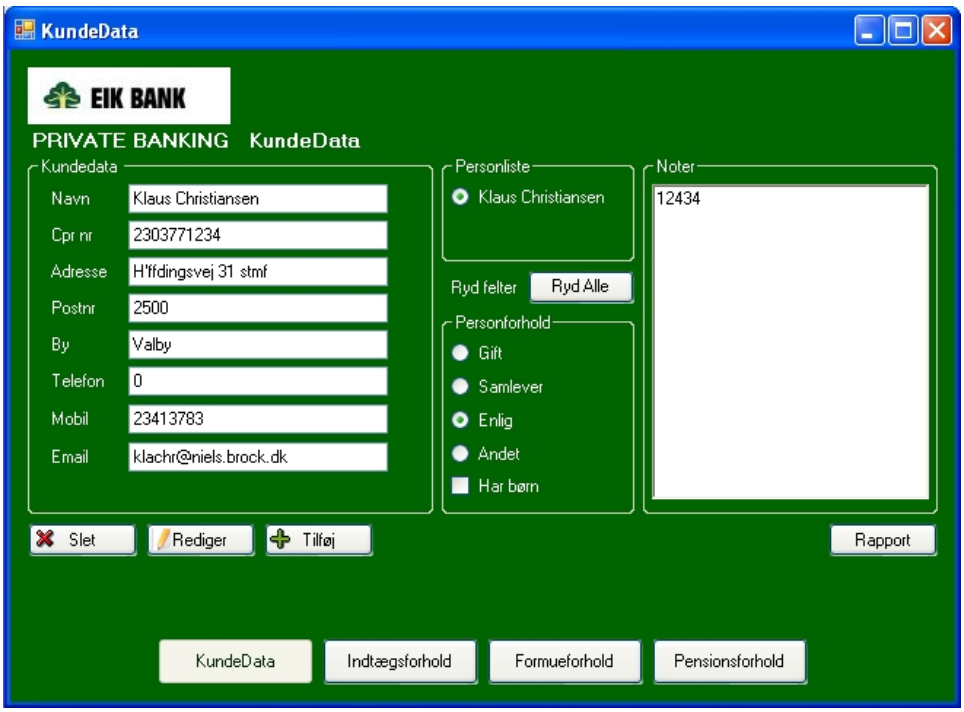
Tidligere session, Sessionsvalg
Når Tidligere sessioner er valgt og ingen session er valgt ⇒ 'OK' knappen gråtones og disbles.
Når man vender tilbage til sessionsvalgsform ⇒ nusltil alle felter.
Når man skifter fra ny session (hvor OK enables) til tidl session skal OK ⇒ disables.

Kundedata
Ingen kunder i sessionen ⇒ radiobuttons visiblity = false; (gælder ved første load af formen).
Når man trykker på 'RydAlle' knap skal information i alle felter slettes, og radioknapperne til personerne være unchecked.
Når ingen kunde er valgt ⇒ 'Slet' og 'Rediger' gråtones og disables. Se figur 6.3 på side 74.
Når en person vælges på personlisten ⇒ 'Slet' og 'Rediger' optones og enables. Se figur 6.4 på side 74.
Når der er 2 person i forholdet ⇒ 'Tilføj' gråtones og disables.

Indtægtsforhold
Når ingen person er valgt ⇒ 'Slet', 'Rediger' og 'Tilføj' gråtones og disables.
Når en person vælges ⇒ 'Tilføj' optones og enables.



Figur 6.3: KundeData: ingen kunde valgt.



Figur 6.4: KundeData: en kunde er valgt.

Når en indtægt vælges ⇒ 'Slet' og 'Rediger' optones og enables.

Formueforhold
Når ingen person er valgt ⇒ 'Slet', 'Rediger' og 'Tilføj' gråtones for både 'Aktiver' og 'Passiver'.
Når en person vælges ⇒ 'Tilføj' optones og enables for både 'aktiver' og 'passiver'.
Når et aktiv vælges ⇒ 'Slet' og 'Rediger' optones og enables.
Når et passiv vælges ⇒ 'Slet' og 'Rediger' optones og enables.

Pensionsforhold
Når ingen person er valgt ⇒ 'Slet', 'Rediger' og 'Tilføj' gråtones og disables.
Når en person vælges ⇒ 'Tilføj' optones og enables.
Når en pension vælges ⇒ 'Slet' og 'Rediger' optones og enables.

Kommentar til brugergrænsefladen

Noget af det første der falder os, som udviklere, i øjnene er hvor svært det er at se forskel på om en knap er klikbar eller ikke klikbar. Grunden til dette skal finde i farvevalget, hvor baggrunden er mørke grøn og knapperne er grå. Et andet problem er at vi har ladet Visual Studio tage sig af forskellene mellem en klikbar og en ikke klikbar knap. En ting vi kunne havde gjort for at forstærke budskabet om hvorvidt en knap er klikbar eller ej, er at vi kunne havde gjort teksten på den pågældende knap mindre tydelig. Dette kunne vi gøre ved at vælge en tekstfarve som ligger tæt på knappens baggrundsfarve. Samtidig kunne vi havde valgt farverne på knapper sådan at de næsten gik i et med skærmens baggrundsfarve. Dette ser vi dog ikke som værende nødvendigt med de knapper som har ikoner. Disse ikoner har vi i to versioner, en med farve til når knappen er klikbar og en uden farver til når knapperne ikke er klikbare.

6.3 Design patterns

Da der foreligger et krav om at vores system skal være modulært og udvidelsesvenligt, er det nærliggende at se på om vi kan indkorporere design patterns i systemarkitekturen. Design patterns går ud på at genkende mønstre i udviklingsfasen og så lave en løsning som er så generel at den kan genbruges næste gang vi støder på dette mønster.

Der findes mange forskellige design patterns som tager sig af forskellige problemområder, på forskellige måder. Man kan bruge disse forskellige måder at tackle problemer og/eller opgaver på, til at kategorisere design patterns i 3 kategorier. De konstruerende, de strukturelle og de handlingsorienterede patterns.

De konstruerende patterns omhandler hvordan man opretter objekter. De strukturelle patterns tager sig af arkitekturen i designet. De handlende patterns fokuserer på hvordan man behandler objekter og hvordan man designer fremtidssikrede applikationer med videre.

Design patterns giver flere fordele:

- Genbrugbarhed af gode løsninger.
- Skaber fælles forståelse og kommunikation mellem udviklere.
- At tænke problemløsning på et højere og mere abstrakt plan.
- Til at beslutte om vi har det rigtige design eller blot et der virker.
- Gør koden nemmere at ændre.
- Gør designet mere modulært.
- Gør koden mere flexibel.
- Gør kode og design mere elegant.

6.3.1 Model View Controller

Vi vil i dette projekt kigge nærmere på det design pattern der hedder MVC[1]. Formålet med dette pattern er at dele et systems ansvarsområder op i tre dele, på en sådan måde at de tre dele bliver uafhængige af hinanden. Det er her, når det kommer til udvidelsesvenlighed, at MVC bliver brugbart. Det er modellagets ansvar at styre behandling af data, databaseadgang og forretningslogik. Det data der bliver sendt fra modellaget skal være brugergrænseflade uafhængigt, hvorved så mange forskellige views som mulig kan bruge modellen. Kontrollaget har ansvaret for at reagere på forandringer i brugergrænsefladen (input fra brugeren). Kontrollaget giver modellaget besked om at den skal ændre sig og giver besked til view om at der er sket

[1]Model View Controller

ændringer i modellen, sådan at view skal opdatere sig. Det er viewlagets ansvar at præsentere data for brugeren, det vil sige at view står for alt hvad der har at gøre med 'look and feel'. Med 'look and feel' menes at det er view der står for at formatere data inden det præsenteres for brugeren.

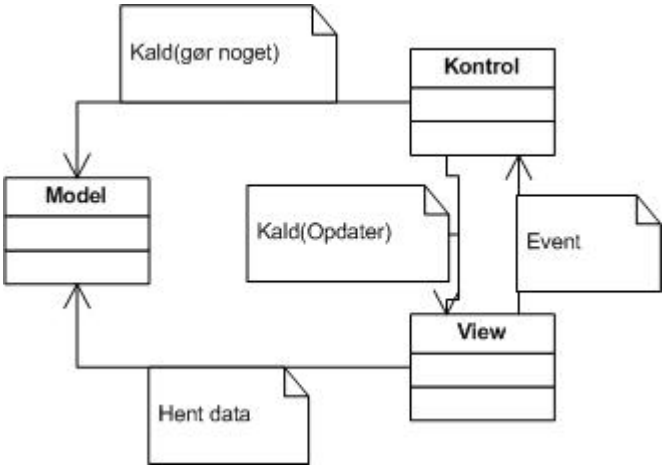
Der er to måder at anskue MVC på, hvor den ene er en aktiv model og den anden er en passiv model. I den passive model ændrer modellaget sig kun når kontrollaget giver besked om det. Dette medfører at modellaget har lav kobling med view og kontrol, hvilket betyder at modellaget ikke ved at de eksisterer. I den aktive model er der andre end view og kontrol som kan ændre modellaget, dette kunne være andre modellag, hvilket gør at modellaget bliver nødt til at give bedsked til view om at den har ændret sig. I dette tilfælde er det nærliggende at gøre brug af at et andet design pattern, nemlig Observer, også kaldet Subscriber/Publisher. Ideen er at view tilmelder sig de modeller de gerne vil have besked fra når der sker ændringer i dem. Modellaget giver dermed besked til alle på listen og viewlaget kan nu hente de nye data.

Hele formålet med design patterns er at generalisere sine løsninger og derigennem at kunne genbruge sin kode og de erfaringer man har med at bygge arkitektureren. Der er ingen grund til at skulle opfinde den dybe tallerken for hver gang man skal strukturere et system.

Det vi vil opnå med MVC er at gøre vores system udvidelsesvenligt. Det smarte er at vi kan ændre forretningslogikken uden at skulle ændre i kontrol og viewlaget.

Et eksempel på brugen af MVC.

```
public partial class KundeData : Form
{
  KundeKontrol kundeKontrol = new KundeKontrol();
  Session session = new Session();

  private void rdoPerson1_CheckedChanged(object sender,
      EventArgs e)
  {
     kundeKontrol.getEvent(sender);
  }

  public void VisKunde(Kunde k)
  {
    Kunde k = session.GetKundeKontainer().GetKunde(0);
    this.txtNavn.Text = k.Navn;
    this.txtCpr.Text = "" + k.CPR;
    this.txtAdresse.Text = k.Adresse;
    this.txtPostnr.Text = "" + k.PostNr;
    this.txtBy.Text = session.GetBy(k.PostNr);
    this.txtTelefon.Text = "" + k.Telefon;
    this.txtMobil.Text = "" + k.Mobil;
    this.txtEmail.Text = k.Email;
```



Figur 6.5: MVC diagram.

```
    this.chkHarBoern.Checked = k.HarBoern;

    if (k.CivilStatus == 1)
      this.rdoStatusGift.Checked = true;
    else if (k.CivilStatus == 2)
      this.rdoStatusSamlever.Checked = true;
    else if (k.CivilStatus == 3)
      this.rdoStatusEnlig.Checked = true;
    else
      this.rdoStatusAndet.Checked = true;
  }
}

public class KundeKontrol
{
  public void getEvent(object sender)
  {
    if (SenderRdo.Name == "rdoPerson1" && kundeForm.
        rdoPerson1.Checked)
    {
      if (session.GetKundeKontainer().GetAntalKunder() >=
          1)
      {
        kundeForm.VisKunde(k);
      }
    }
  }
}
```

Når brugeren trykker på en knap på skærmen, sender view et eventobjekt videre til kontrollaget, som får modellaget til at udføre forretningslogikken.

Når modellaget så er færdig giver kontrol laget besked til view om at den skal opdatere sine data.

Et af de problemer vi har haft med MVC, går ud på at placere ansvar mellem kontrollaget og viewlaget. Det er som sagt view der står for at formatere og præsentere data, og det er kontrollaget der bestemmer hvad der skal vises når det kommer til knapper og andre elementer på skærmen. Man kan sige at view bestemmer hvordan data vises og kontrollaget bestemmer hvornår data vises og hvilke data der vises. Denne balance kan dog gradbøjes og vi oplevede situationer hvor man kunne diskutere hvor det er mest hensigtsmæssigt at placere disse ansvarsområder, alt efter hvilken kode vi stod med.

Det er også kontrollagets ansvar at fortælle modellaget at den skal ændre sig, men det kan til tider være svært udelukkende at holde sig til brugen af kontrol laget. Dette gælder specielt når view skal præsentere noget som er dynamisk og et element på skærmen har betydninger for et andet element. Det betyder at view skal bruge data, som view kun kan få ved at bede modellaget om at lave beregninger. Da vi startede med at kode MVC var vores intention at view kun skulle tilgå modellen med 'get' metoder. Vi har så senere være nød til at ændre denne opfattelse, i retning af at view også må bede modellen om at lave beregninger. Det er dog stadigt kun kontrol som må bede modellen om at ændre sig. Hele denne diskussion bunder i disciplin, og jo mere disciplineret og konsekvent vi koder systemet des nemmere bliver koden at gennemskue, både for os og for andre udviklere.

Til emnet om design patterns og specielt til MVC har vi gjort brug af følgende kilder: [8], [6] og [3].

6.3.2 Singleton

Singleton er et andet design pattern som vi, i modularitetens navn, kan gøre brug af. Ideen med dette pattern er at uanset hvor mange gange man instantierer et objekt, som er kodet som en singleton, så bliver der i memory kun oprettet et objekt ad gangen. Database-tilgangen er en oplagt klasse at implementere som singelton. En fordel ved dette pattern er at der ikke bliver brugt nær så meget tid på at oprette objekter på heapen.

Det kunne se sådan her ud.

```
Class DBAccess{
  private static DBAccess dbInstance;
  public static DBAccess GetDBInstance()
  {
    if (dbInstance == null)
      dbInstance = new DBAccess();
    return dbInstance;
  }
```

```
  private DBAccess()
  {
    string stiTilDB = "C:/Documents and Settings/ideas/
        Skrivebord/HovedOpgave/Test/Accesstestdb/testdb.mdb"
        ;
    string ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;
        Data Source=" + stiTilDB;
    myConnection = new OleDbConnection(ConnString);
    myConnection.Open();
  }
}

public static void main(String[] args)
{
  DBAccess.getDBInstance().Udfør et eller andet();
}
```

Ved at gøre konstruktøren privat forhindrer vi, at der kan oprettes flere objekter af typen DBAccess. Dermed er man tvunget til at bruge metoden GetDBInstance som, via if-sætningen, sikrer at der til enhver tid kun eksisterer ét DBAccess objekt.

Kapitel 7

Test

En del af opgaven i forbindelse med udviklingen af et IT-system handler om at sikre kvaliteten af det udviklede produkt. Med kvalitet tænkes der på flere aspekter. Et aspekt er for eksempel sikkerheden i systemet, hvilket ikke er uvæsentlig i vores projektsammenhæng. Et andet aspekt er rigtigheden af de beregninger systemet laver og om de udviklede algoritmer rent faktisk udfører den opgave de er konstueret til at udføre. Man kan ligeledes undersøge og teste om systemet opfylder den ønskede grad af brugervenlighed og fleksibilitet, som dog er lidt blødere emner der kan være svære at give et konkret svar på. Hvis man ikke undersøger disse, og andre, kvalitetsmæssige aspekter under udviklingsfasen, risikerer man at systemet i sidste ende ikke er brugbart. Denne kvalitetssikring opnås primært gennem forskellige former for tests af systemet, både af systemet som helhed og af de involverede delsystemer.

For at kunne teste et specifikt område af et system, er det nødvendigt at planlægge testsituationen og gøre sig klart hvad det er man vil teste og hvordan man vil teste det. Det giver typisk mest mening at teste både indenfor og udenfor normalområdet af det data systemet kan repræsentere. Hvis en funktion for eksempel ikke skal kunne kan behandle negative heltal, er det hensigtsmæssigt at teste om den så kan håndtere positive heltal, men bestemt også at teste på hvad der sker hvis funktionen mod forventning skulle modtage et negativt heltal. Der skal, så vidt muligt, tages højde for uventede situationer og fejlbehandling af disse, for at undgå at systemet i yderste konsekvens fejler og/eller går ned.

7.1 Testmodeller

Der ligger meget teori og forberedelse til grund for at kunne udføre tests. Da testemnerne kan være ganske forskellige, er det nødvendigt at have forskellige testmodeller til at kunne dække de forskellige test forhold.

Whitebox er en testmodel der bruges til at teste enkelte funktioner i

systemet. Man dykker altså ned i systemets kode og udvælger typisk dele af denne som testmateriale. Kriterierne for om en funktion består en whitebox test ligger i de pre- og postconditions der er opstillet for funktionen. Whitebox tester altså om pre- og postconditions bliver overholdt.

Blackbox ser, som navnet antyder, systemet som en lukket enhed der skal testes, uden kendskab til hvordan systemets interne dele fungerer. Denne testmodel giver input til et givent system og tager derefter det output systemet leverer, som resultat. Med 'et system' menes ikke nødvendigvis en hel og færdig applikation, men kan også være et delsystem.

7.2 Testsession

Test af applikationen handler ikke kun om test af koden, men også om test af andre områder som for eksempel brugergrænseflade. Efter at applikationen er kodet, bør den testes i dens tilsigtede miljø, for at se om den lever op til de krav vi har formuleret i samarbejde med EIK Bank. Kravene er målbare og består i, at den tid indtastningen af en typisk kundes stamdata tager, skal nedbringes ved hjælp af vores applikation. På nuværende tidspunkt og med nuværende arbejdsgang ligger indtastningstiden på ca. 30-60 min. for en gennemsnitskundes datasæt. Kravet er formuleret sådan, at dette skal forbedres efter 5-10 forsøgsrunder med applikationen, ned til i omegnen af 15-30 min, se tabel 3.1 på side 26.

7.2.1 Test af applikationen

For at teste vores applikation vil vi afholde en testsession. Der er dog nogen ting vi bør overveje og planlægge inden vi afholder denne. Vi bør overveje valg af testsubjekter, hvordan testen udføres og hvilken hypotese som skal testes af. Hypotesen bliver formuleret på tilsvarende måde som ovenfor.

- Hypotese: Vores applikation kan hjælpe til at bringe indtastningstiden ned til i omegnen af 15-30 min.

Den ideelle test

Miljøet der testes i bør være så tæt på applikationens tilsigtede miljø som muligt. Dette inkluderer selvfølgelig brugerne, som bør have de samme færdigheder og vidensniveau, som de brugere, der skal arbejde med applikationen i fremtiden. I en ideel testsession vil vi have, de anbefalede, minimum 10 testpersoner, se kilde[5]. Vi kalder dem fremover testbrugere, istedet for testpersoner, da det er applikationen og ikke selve personerne der skal testes.

Vi vil foretage opmålinger af, hvor langt tid det tager at indtaste en kundes data, med rådgivernes nuværende arbejdsgang. Dette vil foregå i deres vante arbejdsomgivelser. En gruppe på minimum 10 testbrugere indtaster

datasæt af gennemsnitsstørrelse på vores system. Efter 5 til 10 forsøg findes den bedste tid, og så har vi et mål for tiden det tager at indtaste datasættet. En sammenligning mellem tiderne for det gamle og vores nye system vil nu være muligt. Det samme vil en vurdering af effekten af det nye system.

Vores testsession

Hvordan blev vores testsession så egentlig afholdt? På trods af at vi er så heldige at have tilgang til de selvsamme brugere, som rent faktisk kommer til at benytte systemet i fremtiden, så har vi kun 2 testbrugere og ikke de anbefalede minimum 10. Dette kommer vores testresultater naturligvis til at lide under rent statistisk set. På dagen for testsessionen viser det sig uheldigvis, at antallet af testbrugere er reduceret til én, på grund af sygdom.

Vi afsatte en time til testsessionen, fordelt på $\frac{1}{2}$ time til selve udførslen af testen og $\frac{1}{2}$ time til uformel snak om sessionen. Vi fordelte roller mellem os. Der var en til at byde velkommen og efterfølgende være observatør, endnu en observatør og en facilitator. Vi benyttede et tidligere udleveret datasæt, som testbrugeren skulle køre igennem. Dette datasæt bestod af materiale fra en rigtig kundesag. Testbrugeren blev stillet to scenarier. Først skulle testbrugeren finde kundens stamdata i datasættet og indtaste dem, dernæst skulle indtægtsforholdene findes og indtastes. Herefter stillede vi en håndfuld improviserede tillægsspørgsmål, hvor testbrugeren blev bedt om at redigere og slette henholdsvis en indtægt og en person, for at komme rundt i de sidste dele af systemet. I samtalen med EIK Bank rådgiveren bagefter, ville vi fokusere på systemets brugervenlighed og effektivitet, og tage noter imens.

7.2.2 Resultat af testsessionen

Sessionen forløb fint og varede præcist en time som planlagt. Tidsmæssigt tog det testpersonen 11 minutter og 24 sekunder at indtaste henholdsvis kundedata, indtægtsforhold og det tog yderligere 3 minutter og 31 sekunder komme igennem rediger og slet funktionerne.

Navigeringen rundt mellem de forskellige forme gik faktisk ret godt. Der var dog lidt forvirring omkring 'tilføj' knappen. Under oprettelsen af en kunde gik det fint nok med at trykke på 'tilføj' knappen, men da der skulle oprettes en indtægt, gik der længere tid inden testbrugeren fandt ud af, at man skulle markere en kunde, inden man kunne trykke på 'tilføj' knappen. Herefter var der skabt forståelse for, at en kunde skulle markeres før der kunne indtastes data. 'Rediger' og 'slet' indtægter virkede ikke helt intuitivt for testbrugeren og krævede lidt hjælp at komme igennem. Det skal så nævnes at det var første gang testbrugeren så og anvendte systemet, udover at have set prototyperne. Testbrugeren fik ikke tid til først at lære systemet at kende. Sessionen begyndte med det samme.

Der var også lidt forvirring i begyndelsen omkring, at en kunde hører

sammen med en session. Altså at der ikke kan eksistere en kunde uden at der eksisterer en session. Dette kunne der sandsynligvis rettes op på ved at kalde en 'session' for en 'sag' i stedet for, har vi erfaret efterfølgende. Bortset fra ovennævnte problemer navigerede testbrugeren ret godt rundt i programmet.

Under samtalen bagefter kom vi lidt ind på de tidsmæssige aspekter i systemet, brugergrænsefladen og systemet generelt. Vi fandt frem til en hel del forslag til hvordan applikationen kan forbedres. Både nogle vi var klar over og nogle der var nye. Her er noget af det vi fandt frem til:

Kundedata

- På nuværende tidspunkt stiller programmet krav om at der under oprettelsen af en kunde skal indtastes følgende stamdata: 'CPR nummer', 'postnummer', 'mobilnummer' og 'telefonnummer', mens 'Navn', 'adresse' og 'email' kan udelades. For fleksibilitetens skyld vil det være bedre hvis det kun var 'Navn' og 'CPR nummer' der var nødvendige at indtaste.

- I den situation hvor der skal tilføjes en person nr. 2, burde systemet være skruet sådan sammen, at efter der er blevet trykket på tilføj, skulle alt information i textfelterne der er indtastet bevares. Kun 'CPR nummer' og 'Navn' skal ryddes.

Indtægtsforhold

- Der ønskes helt overordnet en opdeling af ansvaret mellem popup´en og indtægtsformen, på en sådan måde, at indtægtsformen giver overblikket og popup´en giver detaljerne.

- For testbrugeren virkede det mere logisk at have knapperne til 'tilføj', 'slet' og 'rediger' på popup-formen.

- Sum på popup-formen. Således at der er en samlet sum for den enkelte kategori, f.eks. aktier.

- Totalsum på hovedformen. Totalsum for samtlige kategorier på formen, altså alle indtægter.

- Når der tilføjes en ny indtægt skal den ikke indsættes nederst i treeview'et på indtægtsformen, men følge samme rækkefølge af indtægter som på popup'en. Treeview bør altså være sorteret.

- På popup'en ønskes det, at alle posteringerne under den enkelte kategori skal listes på én gang og at de skal listes i rækker og kolonner.

- Kundens navn på popup skærmen vil måske være en god ide.

Generelt

- Fortrydknap: Testbrugeren efterspurgte en 'fortrydknap', som han kendte fra andre programmer. Han synes dog ikke problemet er så stort, da der ikke er nogen reel risiko for at miste data. Al informationen der skal indtastes, ligger alligevel i kundens datasæt og kan indtastes igen.

- En advarsel når brugeren vælger 'slet', som beder om en bekræftigelse af sletningen.

- Tusindtalsseperatorer for at tydeliggøre beløbenes størrelser.

Vi fik også input til vores grafiske brugergrænseflade. Testbrugeren tillagde ikke ikonerne på 'tilføj', 'slet' og 'rediger' knapperne nogen betydning. Plustegnet på 'tilføj' knappen og krydset på 'slet' knappen gav ikke nogen associationer i retning af at slette og tilføje. Blyanten på 'rediger' knappen gav heller ikke associationer i retning af at kunne redigere. Til gengæld kom testbrugeren med et forslag til et nyt ikon til redigerknappen, nemlig et viskelæder. Ideen om at viskelæderet giver bedre associationer end blyanten, må vi tilslutte os. Til gengæld var der bedre forståelse for farverne grøn og rød, som symboliserede "go" og "stop".

For at vende tilbage til vores hypotese, om at vores system kan nedbringe indtastningstiden fra 30-60 minutter til 15-30 minutter, så mente testbrugeren direkte adspurgt, at man efter 15 minutters indøvelse vil have vænnet sig til det nye system. Vi fik ikke svar på hvorvidt testbrugeren troede at indtastningstiden, ved brug af vores system, vil komme ned på 15-30 minutter, men han gav udtryk for at systemet vil give en forbedring. Vi spurgte ind til hvad grunden til denne forbedring i forhold til den nuværende arbejdsgang kunne være. Testbrugeren var af den opfattelse at det nye system kommer til at agere som en slags huskeliste. Med den nuværende arbejdsgang bruges et eksisterende excelark, som en slags støtte eller skabelon, så en del indtastningsfelter kan genbruges og ny data indtastes. Idet to kunder aldrig er ens og at der kun gemmes relevant information i Excelarket, så opstår problemet med at rådgiveren risikerer at glemmer at indtaste nogle punkter. Vores system sikrer, at der ikke er nogen områder der glemmes. Muligheden for at springe rundt mellem de forskellige områder er bygget ind i applikationen. Huskelistefunktionaliteten, i mangel på et bedre ord, hæmmer altså ikke fleksibiliteten. Selv om det kan lyde som om hypotesen holder, skal man lige huske på de ting som kan så tvivl om rigtigheden af de fundne påstande.

Vi har kun haft én testbruger af systemet, og dette er ikke nok til at fastslå noget med sikkerhed. Det kræver flere testbrugere før det statistiske materiale er godt nok til at kunne konkludere noget. Testbrugeren har heller ikke haft meget tid til at lære systemet at kende, så dataindtastningen vil selvfølgelig kunne foregå en del hurtigere end under denne session. At systemets pensions- og formuedele heller ikke er implementerede, er også med til at give det fundne resultat en vis usikkerhed. Vi kan ikke teste i rådgiverens

UserID (PK)	Name	Address	Telephone	Email
autoincrement	text	text	int	text

Tabel 7.1: Test database

daglige miljø, og dermed tage højde for diverse forstyrrende elementer. Vores testsession fandt dog sted i ret lignende rammer, så vi vurderer det til at være relativt ubetydeligt.

Disse faktorer gør at vi ikke er i stand til at konkludere noget håndfast, men at vi snarere må kalde det en indikation af, at vores system er i stand til at opfylde den stillede hypotese.

7.3 Test af databasetilgang

Denne test har 3 formål.

1. Vi skal skabe forbindelse til databasen.
2. Vi skal teste om vi kan skrive til databasen.
3. Vi skal teste om vi kan læse fra databasen.

Til dette formål skal vi bruge en database, og den ser ud som på følgende figur, se 7.1 på side 86. Til at starte med er databasen tom.

```
class DBAccess
{
    private OleDbConnection myConnection;
    private OleDbDataAdapter myDataAdapter;
    private OleDbCommand myCommand;

    public DBAccess()
    {
        Console.WriteLine("Skriv stien til access databasen"
            );
        String stiTilDB = Console.ReadLine();
        string ConnString = "Provider=Microsoft.Jet.OLEDB
            .4.0;Data Source=" + stiTilDB;
        myConnection = new OleDbConnection(ConnString);
    }

    public DataSet ExecuteQuery(string sql)
    {
        DataSet myDataSet;
        myDataSet = new DataSet();

        try
        {
```

```
            myConnection.Open();
            myCommand = myConnection.CreateCommand();
            myCommand.CommandText = sql;
            myDataAdapter = new OleDbDataAdapter(myCommand);
            myDataAdapter.Fill(myDataSet, "result");
        }
        catch (Exception e)
        {
            myDataSet.DataSetName = e.Message;
            Console.WriteLine(e.Message);
            return myDataSet;
        }
        finally
        {
            myConnection.Close();
        }
        myDataSet.DataSetName = "Result";
        return myDataSet;
    }

    public string ExecuteNonQuery(string sql)
    {
        try
        {
            myConnection.Open();
            myCommand = myConnection.CreateCommand();
            myCommand.CommandText = sql;
            myCommand.ExecuteNonQuery();
        }
        catch (Exception e)
        {
            return e.Message;
        }
        finally
        {
            myConnection.Close();
        }
        return "ok";
    }
}

static void Main(string[] args)
{
  //oprette et DBAccess objekt.
  DBAccess dba = new DBAccess();

  //Indsætter en 'user' i databasen.
  String sql = "insert into Users (Name, Address, Telephone
      , Email) values ('Klaus Christiansen', 'Høffdingsvej
```

```
        31', 23413783, 'kec2@email.dk')";
    //udskriver om det gik godt eller skidt.
    Console.WriteLine("Data blev indsat i db :" + dba.
        ExecuteNonQuery(sql));
    Console.ReadKey();

    sql = "select * from Users";
    //Vælger alle 'users' i databasen.
    DataSet ds = dba.ExecuteQuery(sql);
    //Skriver alle 'users' ud en for en.
    try{
      foreach (System.Data.DataRow r in ds.Tables["result"].
          Rows)
      {
          Console.WriteLine("User ID    : " + r["UserId"].
              ToString());
          Console.WriteLine("Name       : " + r["Name"].
              ToString());
          Console.WriteLine("Address    : " + r["Address"].
              ToString());
          Console.WriteLine("Telephone  : " + r["Telephone"].
              ToString());
          Console.WriteLine("Email      : " + r["Email"].
              ToString());
          Console.WriteLine();
          Console.ReadKey();
      }
    }
    catch(NullReferenceException nre){
      Console.WriteLine("Der er sket en fejl : "+nre.messege)
          ;
    }
}
```

Til at lave denne test vil vi gøre brug af en blackbox test. Vi vil tage udgangspunkt i at der findes en database som før beskrevet, se tabel 7.1 på side 86. Skulle denne database ikke være tilgængelig, vil testen fejle. Dette vil resultere i at følgende bliver udskrevet til skærmen:

```
Data blev indsat i db : 'stien' kan ikke findes.
```

Hvor stien er den absolutte man har skrevet. Skulle vi derimod forsøge af udskrive data som ikke findes i databasen vil programmet smide en exception som vil bliver grebet, og fejlen vil blive udskrevet.

7.3.1 Selve testen

Det første der sker er at vi bliver bedt om at indtaste den absolutte sti til databasen. I vi vores tilfælde skriver vi 'C:/Documents and Settings/ideas/Skrivebord/HovedOpgave/Test/Access testdb/testdb.mdb'.

SQL streng 1 = "insert into Users (Name, Address, Telephone, Email) values ('Klaus Christiansen', 'Høffdingsvej 31', 23413783, 'kec2@email.dk')"
SQL streng 2 = "select * from Users"
Som resultat giver denne test følgende output: Se tabel 7.2 på side 89.

Indput	Forventet output	Faktisk output	Konklusion
SQL streng 1	Data blev indsat i db : ok	Data blev indsat i db : ok	√
SQL streng 2	User ID : > 0	User ID : 56	√
	Name : Klaus Christiasen	Name : Klaus Christiasen	√
	Address : Høffdingsvej 31	Address : Høffdingsvej 31	√
	Telephone : 23413783	Telephone : 2341783	√
	Email : kec2@email.dk	Email : kec2@email.dk	√

Tabel 7.2: Resultat af databasetest.

7.4 Test af gem funktioner

Vi skal vise at gem-algoritmen virker. Algoritmen kan ses i afsnit 5.2.3 på side 51. Nedenfor ses et udsnit af 3 klasser, hvor kun de interessante metoder er med. Vi vil vise at algoritmen virker når der oprettes flere aktiver i databasen. Vi har ovenover set at tilgangen til databasen virker. Den opmærksomme læser vil se at der er forskel på DBAccess ovenover og de kald der bliver lavet nedenunder til selv samme DBAccess klasse. Dette skyldes at ovenstående test er lavet før vi implementerede design patternet 'Singleton'. Der står mere om dette design pattern i afsnit 6.3.2 på side 79.

7.4.1 Forklaring af kode

Det første af de 3 klasseudsnit der ses nedenfor, er klassen 'Aktiv'. Denne klasse er en superklasse, hvilket vil sige at andre klasser kan arve fra den. Den har blandt andet metoden *GemAktiv()* som tager sig at de indledende øvelser når et aktiv skal gemmes i databasen. Det næste udsnit er af klassen 'Aktie' som er en subklasse, da den arver fra 'Aktiv'. Den har også en metode der hedder *GemAktiv()* og den tager sig af at gemme det som er specifikt for en 'Aktie'. Det sidste er et udsnit af klassen 'AktivKontainer'. Udsnittet viser at når en kontainer af typen 'Aktiv' instantieres, så gemmes 6 forskellige 'Aktiver'.

For at testen skal virke er der nogle ting vi må antage. For det første skal der være to tupler i tabellen 'Kunde'.

Kunde	
KundeID	Navn
13	Default
2	TueT

Vi har desuden brug for at antage følgende om tabellen 'Aktiv'.

Aktiv	
AktivID	KundeID
346	13

Til sidst antager vi at tabellerne Aktie, BankIndestaaende, FastEjendom med flere, er tomme. Når der bliver indsat en tupel i en af disse tabeller, så starter deres ID hver især fra 1, og hvergang der indsættes en ny tupel så bliver der lagt en til. Dette betyder at tabellerne sagtens kan have samme ID.

Aktie	
AktieID	AktivID

BankIndestaaende	
BankIndeStaaendeID	AktivID

Det skal for en god ordens skyld nævnes at det kun er et udsnit af kolonnerne der er vist i tabellerne.

Første kald i *Aktiv.GemAktiv()* giver os $kundeid = 13$ og $AktivID = 346$. Så opretter vi en tupel i tabellen 'Aktiv' som også peger på 'Kunde' med $KundeID = 13$. Denne nye tupel får nu $kundeid = 13$ og $AktivID = 347$. Så opdatere vi 'Aktiv' tabellen og sætter $KundeID = 2$ som er ID'et på vores kunde, hvor $AktivID = 346$. Nu tager *Aktie.GemAktiv()* over og opretter en tupel i 'Aktie' som peger på vores $AktivID = 346$. Til sidst henter vi det nye *AktieID* som nu er 1. Aktien får $ID'et = 1$.

Da denne serie kald startede var det *AktivID*, vi kunne bruge, 346. Nu når der er indsat en 'Aktie' er *AktivID* et blevet $346 + 1 = 347$. Så hver gang denne algoritme kører, bliver *AktivID* én større og på den måde kan vi blive ved med at indsætte forskellige 'Aktiver' i databasen.

```
public abstract class Aktiv
{
  public virtual void GemAktiv()
  {
    // Henter tuplen, i tabellen aktiv, med det id som
        peger på kunden med navnet 'Default'
    string sql = "SELECT aktivid, kunde.kundeid FROM aktiv
        inner join kunde on aktiv.kundeid = kunde.kundeid
        WHERE navn='Default'";
```

```
    DataSet myDataSet = DBAccess.getDBInstance().
        ExecuteQuery(sql);
    int emptyAktivID = 0;
    int defaultKundeID = 0;
    try
    {
      emptyAktivID = Convert.ToInt32(myDataSet.Tables["
          Result"].Rows[0][0].ToString());
      defaultKundeID = Convert.ToInt32(myDataSet.Tables["
          Result"].Rows[0][1].ToString());
    }
    catch (NullReferenceException nre)
    {
      Console.WriteLine("Aktiv.GemAktiv 1 : " + nre.Message
          );
    }
    //Indsætter en ny tupel der nu får det højeste id (id
        +1)
    sql = "INSERT INTO aktiv (kundeid) VALUES (" +
        defaultKundeID + ")";
    DBAccess.getDBInstance().ExecuteNonQuery(sql);

    //Opdaterer den hentede tupels kundeid til den
        pågældende kunde
    sql = "UPDATE aktiv SET kundeid=" + KundeID + " WHERE
        aktivid=" + emptyAktivID;
    DBAccess.getDBInstance().ExecuteNonQuery(sql);
    this.AktivID = emptyAktivID;
  }
}

public class Aktie : Aktiv
{
  public override void GemAktiv()
  {
    //Opretter plads i tabellen Aktiv
    base.GemAktiv();
    //Indsætter en aktie i tabellen aktie, med relation til
        det ovenfor oprettede aktiv
    string sql = "INSERT INTO aktie (aktivid, selskab,
        antal, kurs, friemidler, pensionsmidler) values (" +
         base.AktivID + ", '" + selskab + "', " + antal + ",
        " + kurs + ", " + frieMidler + ", " +
        pensionsMidler + ")";
    DBAccess.getDBInstance().ExecuteNonQuery(sql);

    //Henter det pågældende AktivID
    sql = "SELECT aktieid FROM aktie WHERE aktivid=" + base
        .AktivID;
```

```
    DataSet myDataSet = DBAccess.getDBInstance().
        ExecuteQuery(sql);

    try
    {
      aktieID = Convert.ToInt32(myDataSet.Tables["Result"].
          Rows[0][0].ToString());
    }
    catch (NullReferenceException nre)
    {
      Console.WriteLine("Aktie.GemAktie : " + nre.Message);
    }
  }
}

public class AktivKontainer
{
  public AktivKontainer(int kundeID_)
  {
    kundeID = kundeID_;
    aktiver = new List<Aktiv>();

    Aktie a = new Aktie(0,2,0,"TestAktie",100,100,false,
        false);
    a.GemAktiv();
    AndreAktiver aa = new AndreAktiver(0, 2, 0, 100);
    aa.GemAktiv();
    Bankindestaaende b = new Bankindestaaende(0, 2, 0, 100)
        ;
    b.GemAktiv();
    FastEjendom f = new FastEjendom(0, 2, 0, 100, 100, 100,
        100);
    f.GemAktiv();
    Obligation o = new Obligation(0, 2, 0, "TestObligation"
        , 100, 100, false, false);
    o.GemAktiv();
    SelvstaendigVirksomhed s = new SelvstaendigVirksomhed
        (0, 2, 0, 100, 100, 100);
    s.GemAktiv();
  }
}
```

Når testen er kørt igennem har tabellerne følgende data.

Kunde	
KundeID	Navn
13	Default
2	TueT

Aktiv	
AktivID	KundeID
346	2
347	2
348	2
349	2
350	2
351	2
352	2
353	2
354	2
355	2
356	2
357	2
358	13

Aktie	
AktieID	AktivID
1	346
2	352

BankIndestaaende	
BankIndeStaaendeID	AktivID
1	348
2	354

Nedenfor ses de SQL-kald der ligger til grund for tabellerne.

```
SELECT aktivid, kunde.kundeid FROM aktiv inner join kunde
    on aktiv.kundeid = kunde.kundeid WHERE navn='Default'
INSERT INTO aktiv (kundeid) VALUES (13)
UPDATE aktiv SET kundeid=2 WHERE aktivid=346
INSERT INTO aktie (aktivid, selskab, antal, kurs,
    friemidler, pensionsmidler) values (346, 'TestAktie',
    100, 100, False, False)
SELECT aktieid FROM aktie WHERE aktivid=346

SELECT aktivid, kunde.kundeid FROM aktiv inner join kunde
    on aktiv.kundeid = kunde.kundeid WHERE navn='Default'
INSERT INTO aktiv (kundeid) VALUES (13)
UPDATE aktiv SET kundeid=2 WHERE aktivid=348
INSERT INTO bankindestaaende (aktivid, beloeb) values (348,
     100)
SELECT bankindestaaendeid FROM bankindestaaende WHERE
    aktivid=348

SELECT aktivid, kunde.kundeid FROM aktiv inner join kunde
    on aktiv.kundeid = kunde.kundeid WHERE navn='Default'
INSERT INTO aktiv (kundeid) VALUES (13)
```

```
UPDATE aktiv SET kundeid=2 WHERE aktivid=352
INSERT INTO aktie (aktivid, selskab, antal, kurs,
    friemidler, pensionsmidler) values (352, 'TestAktie',
    100, 100, False, False)
SELECT aktieid FROM aktie WHERE aktivid=352

SELECT aktivid, kunde.kundeid FROM aktiv inner join kunde
    on aktiv.kundeid = kunde.kundeid WHERE navn='Default'
INSERT INTO aktiv (kundeid) VALUES (13)
UPDATE aktiv SET kundeid=2 WHERE aktivid=354
INSERT INTO bankindestaaende (aktivid, beloeb) values (354,
     100)
SELECT bankindestaaendeid FROM bankindestaaende WHERE
    aktivid=354
```

Kapitel 8

Fremtiden

8.1 Fremtidige udvidelsesmuligheder for systemet

- Mobilitet. EIK Bank har udtrykt ønske om at rådgiveren kan tage applikationen med ud til kunden, på en bærbar computer. Denne funktionalitet er dog blevet nedprioriteret i forhold til den basale funktionalitet og den tidsbegrænsning projektet er omfattet af. Dermed er den blevet en mulig fremtidig udvidelse. Denne funktionalitet indebærer dog nogle problematikker med synkronisering af data med den centrale database i EIK Bank. Vi ser to mulige løsninger på dette.

  Hvis rådgiveren skal på besøg hos en kunde er en mulig løsning at medbringe data fra den centrale database. Rådgiveren skal altså forberede applikationen ved at hente kundens data ned, inden han frakobler den bærbare computer fra EIK Banks netværk og tager den med. Yderligere vil rådgiveren, når han vender tilbage til EIK Banks netværk med de nye data, skulle tilføje de nye kundeforhold til den centrale database.

  Den anden mulighed for at løse problemet er at rådgiveren medbringer et mobilmodem ud til kunden og kan koble sig til internettet. Derfra kan han, for eksempel ved hjælp af en webservice, koble sig direkte til den centrale database. Vi har ikke undersøgt aspekterne omkring hverken sikkerheden eller implementeringsomkostningerne af sådan en løsning.

  I forbindelse med denne funktionalitet er der yderligere leget med tanken om muligheden for at rådgiveren kan medbringe en bærbar printer og afslutte besøget med at printe resultatet af rådgivningen ud til kunden med det samme. Dette mener flere medarbejdere ville øge graden af professionalisme og være en rigtig smart funktionalitet.

- Integration med EIK Banks eksisterende systemer. Fra vores side var der i begyndelsen tanker om muligheden for at lade vores applikation

hente kundedata fra EIK Banks nuværende kundesystem hos deres datacentral. Dette var dog før vi indså hvor lukket EIK Banks system er, hvilket jo er mere end naturligt nok, når man tænker på hvor kritiske de data banken behandler er. Fra bankens side har der også været tale om sammenkobling med eksisterende systemer, som en væsentlig udvidelsesmulighed.

- Øget fleksibilitet. Der har været talt om muligheden for at brugerne af systemet kan bestemme om der kan tilføjes flere indtastningsfelter, som for eksempel telefonnumre og emails. Dette kunne gøres i en konfigurationsdel i systemet, hvor antallet af indtastningsfelter og hvilke indtastningsfelter der ønskes nemt kan ændres. Dette kræver dog at databasen er forberedt for denne udvidelse og der er mulighed for at den kommer til at indeholde mange NULL-værdier.

- Mulighed for at udføre fremskrivninger og lave beregninger. Med den klasseopdeling applikationens arkitektur er bygget på, er det oplagt at udvide programmets funktionalitet til også at kunne udføre beregninger. Selvom kravet om dette er nedprioriteret, er det stadig en væsentlig funktionalitet for systemet, som har været tiltænkt fra begyndelsen.

- Optimering af notesystemet. Notesystemet er et punkt hvor der ligger mange muligheder for at udvide og øge funktionalitet. For det første kunne man overveje at lave en form for formatering af teksten i notefeltet, hvormed der bliver lavet linieskift efter en note, samt for eksempel en streg til at adskille hver note. Man kunne yderligere indsætte et timestamp, rådgiverens navn med mere. Udover disse forbedringer af det eksisterende notefelt, er det oplagt at se på en mere radikal ændring af notekonstruktionen. Man kunne starte med at lave en klasse der indeholder data til noten. Hermed får man bedre mulighed for at lave udvidet funktionalitet og formatering.

  Man kunne også overveje at lave en popup-form til al notehåndtering, som kan tilgås via en noteknap på hvert skærmbillede i systemet. Hermed samles tilgangen til notesystemet til ét sted. Hvis denne løsning gør noten lidt for adskilt fra sessionen, kunne man lave et uredigerbart notepreview på hvert skærmbillede hvor for eksempel seneste note vises. Yderligere kunne man overveje om noten skal være samlet for sessionen eller om noterne skal være specifik per kunde i sessionen.

- Optimering af sessionsvalget. Vi begyndte med at have dato, samt rådgivers navn, på sessionvalgsformen. Dette er nu efter EIK Banks ønske, ændret til kundenavn istedet. Det er mere vigtigt at kunne se kundens navn end det er at kunne se rådgiverens og datoen.

Det er yderligere et ønske fra EIK Banks side at der er mulighed for at søge blandt oprettede sessioner og kunder. Jo flere sessioner der kommer, des længere og mere uoverskuelig bliver listen, hvormed en søgefunktionalitet vil være hensigtsmæssig at implementere. Dette vil kunne lette arbejdsgangen med at finde frem til en session eller en kunde betydeligt. Da de data der søges på allerede er indlæst i datastrukturen vil søgningen også være forholdsvis hurtig, rent performancemæssigt.

I EIK Banks oplæg til opgaven, ligges der endvidere op til yderligere udvidelsesmuligheder til systemet, se også bilag 10.1 på side 111:

- *Udvidelser til kundeforholdet, som typisk vil manifesteres i form af tabeller i databasen eller anden funktionalitet i form af direkte links til websites.*
  - *Test vedrørende risikoanalyse vedrørende risikoprofil.*
  - *Pensions- og dækningsforhold for de foretrukne samarbejdspartnere/forsikringsselskaber.*
  - *Værdipapirer over foretrukne porteføljer, for eksempel EIK Banks modelportefølje.*
  - *Skatteforhold i foretrukne udlande, for eksempel Frankrig, Spanien, England med flere.*
  - *Gængse realkreditlån, beregningsmotor, omlægning af lån.*
  - *Udregning af konsekvensen af renteudviklinger.*

Kapitel 9

Konklusion

9.1 Konklusion

I forbindelse med kravet om brugervenlighed, har vi studeret en del teori vedrørende design af brugergrænseflader og det at skabe brugervenlighed. Ud fra denne teori har vi reflekteret over visse aspekter.

Den generelle teori om design af brugergrænseflader ligger vægt på brugerinddragelse i analyse- og designprocessen. Dette stemmer godt overens med de elementer af eksperimentiel systemudvikling som vi har anvendt. Brugervenligheden har vi opnået ved at inddrage brugerne i Lo-Fi prototypesessioner og på den måde har brugerne fået mulighed for at få deres terminologi med på brugergrænsefladen.

I forhold til princippet om at vi skal bruge metaforer fra brugernes egen verden og at programmet skal være så intuitivt at eksempelvis fejldialoger ikke er nødvendige, har vi ligeledes medtaget brugernes meninger og ønsker i vores overvejelser. Vi fandt ud af at vi, som udviklere, ikke altid bruger samme metaforer som brugerne. Dette område brugte vi ikke helt nok tid på at analysere, hvorfor vi burde have afholdt en session om emnet med brugerne.

Gestaltlovene, som bygger på menneskers universelle opfattelse af helheder, har vi brugt til at skabe helheder i vores skærmbilleder. Her har vi forsøgt at gruppere relaterede elementer ved siden af hinanden og bruge lukkethed og lighed til at skabe et brugervenligt design. I vores tilfælde har vi kunne gøre brug at flere gestaltlove, hvor vi har kombineret styrkerne fra de enkelte love til at skabe brugervenlighed.

Vi erfarede at der fandtes et program i vores udviklingsmiljø, til at skabe professionelt udseende rapporter, nemlig Crystal Reports. Værktøjet har vist sig komplekst at arbejde med og det største problem vi har haft i dette regi, er at vi ikke kendte værktøjet på forhånd. Vi manglede erfaring med det og havde ikke tid til at sætte os ind i det. Hvis udprintsfunktionaliteten i systemet skal nå et niveau i overensstemmelse med EIK Banks krav,

ville mere erfaring i brugen af Crystal Reports være nødvendig. Vi havde forventninger og ønsker om mere, men på dette punkt må vi dog sande at planlægningen ikke tillod dette. Vi lærte, trods alt, at man kan blive nødt til på et tidspunkt at sige 'stop' og ikke lade problemet ødelægge vores muligheder for at overholde tidsplanen og at udføre de opgaver der venter.

Med hensyn til det professionelle udseende på systemets udprint er vores erfaring at det har været svært at få sat ord på hvad et professionelt look egentlig er. Der har været nøgleord som 'konservativt udseende', logoer, fonte og så videre på banen, men ud fra dette har vi ikke kunne danne os en helt præcis opfattelse. Derfor har vi ikke været gode nok til at stille de rigtige spørgsmål til kunden.

Kravet om overskuelighed og gennemskuelighed overfor kunden, kan vi ikke sige at have opfyldt, da vi ikke har kunne formatere systemets udprint på en overskuelig måde.

På grund af den begrænsede tid vi, i projektet, har haft til rådighed, føler vi at det var den helt rigtige beslutning at inddrage elementer fra eksperimentiel- og agil systemudvikling i processen. Det har, blandt andet ved hjælp af prototyping, betydet at vi er kommet hurtigt og effektivt igang og vi har haft mere tid til udviklingen af produktet, end vi ville have haft hvis vi havde fulgt UPs metodologi. Prototyping har endvidere også været en erfaring for virksomhedens medarbejdere, der både blev taget med i og fik indsigt i processen, og fik mulighed for at tænke en ekstra gang over hvad det egentlig er de har brug for. Ved hjælp af Scrum-morgenmøderne, som giver en løbende opfølgning på processen og det arbejde der udføres, har vi sørget for hele tiden at holde os på rette spor og tilpasse os situationen. Dette har virket godt og effektivt. Hertil har ugeplanlægningen og fredagsbriefingen bidraget yderligere.

Med hensyn til spørgsmålet om brugen af design patterns for at gøre det nye system modulært og udvidelsesvenligt, kan vi konkludere følgende.

Ved at bruge Model View Controller har vi fået delt arkitekturen op på en sådan måde at modellaget er fuldstændig uafhængigt af kontrol- og viewlaget. Dette betyder at vi kan lave nye brugergrænseflader eller ændre i eksisterende uden at det får nogen indflydelse på modellaget. Dermed kan vi se systemet som to dele, nemlig kontrol og view som den ene del, og model som den anden del. Kigger vi nærmere på de dele der indgår i modellaget, så har de to design patterns som vi har brugt, ikke nogen indflydelse på om disse dele i sig selv er modulære og udvidelsesvenlige.

Når design patterns ikke kan opfylde ønsket om modularitet, bliver man nødt til at designe og kode sig ud af problemet. Her kan man konstruere systemets dele med en høj grad af samhørighed og lav kobling. Dette har vi eksempelvis opnået med vores database klasse 'DBAccess'. Hvis vi skifter eller flytter databasen, så skal der kun rettes ét sted i koden. Dermed opfattes

'DBAccess' som et modul.

Det andet design pattern vi har valgt at benytte os af, nemlig Singleton, har ikke bidraget til en højere grad af modularitet og udvidelsesvenlighed. Vi har udelukkende brugt dette af ydelse- og strukturmæssige grunde.

Ved at bruge arv til implementeringen af entiteterne i 'indtægter', 'aktiver', 'passiver' og 'pensioner', sikrer vi at de generelle databehandlinger, som er fælles for alle entiteter, er samlet et sted og kan genbruges når der udvides med flere typer entiteter.

EIK Bank står med andre ord med et system hvor den grundlæggende arkitektur er på plads, i form af MVC strukturen. Dette åbner for EIK Banks muligheder for at udvide systemet. Den grundlæggende funktionalitet i delsystemet indtægtsforhold er lavet, og kan derfor fungere som skabelon for funktionaliteten i formue- og pensionsforhold. Når dette er på plads er systemet ikke langt fra at kunne udvides med fremskrivningsfunktionalitet. Systemet giver yderligere EIK Bank en centraliseret database over de sager de arbejder med i private banking afdelingen. Systemet giver på nuværende tidspunkt ikke mulighed for, på ordentlig vis, at printe en pæn og overskuelig rapport ud til kunden.

Udover dette har EIK Bank fået et indblik i hvad det kræver for at lave et specialiseret system fra bunden, som vi håber kan bruges.

9.2 En tak til EIK Bank

Vi vil gerne benytte lejligheden til at takke EIK Bank for deres store engagement i dette projekt. Vi har som studerende fået stillet kontor, computere og meget andet til rådighed, som vi har kunnet benytte alt det vi har måttet ønske, indenfor bankens åbningstid. Vi har kunnet spørge bankens medarbejdere efter behov og de har stillet sig til rådighed når vi har haft brug for det. Disse forhold gav os et godt grundlag for at kunne fokusere på opgavens problemstillinger, fremfor mere praktiske problemer.

Kapitel 10

Bilag

Klassediagram

I dette afsnit vil vi kort gennemgår vores klassediagram. Diagrammet har et sådan omfang at vi har været nød til at dele det op i mindre bidre, dels for at gøre det muligt at læse og dels for at skabe overblik.

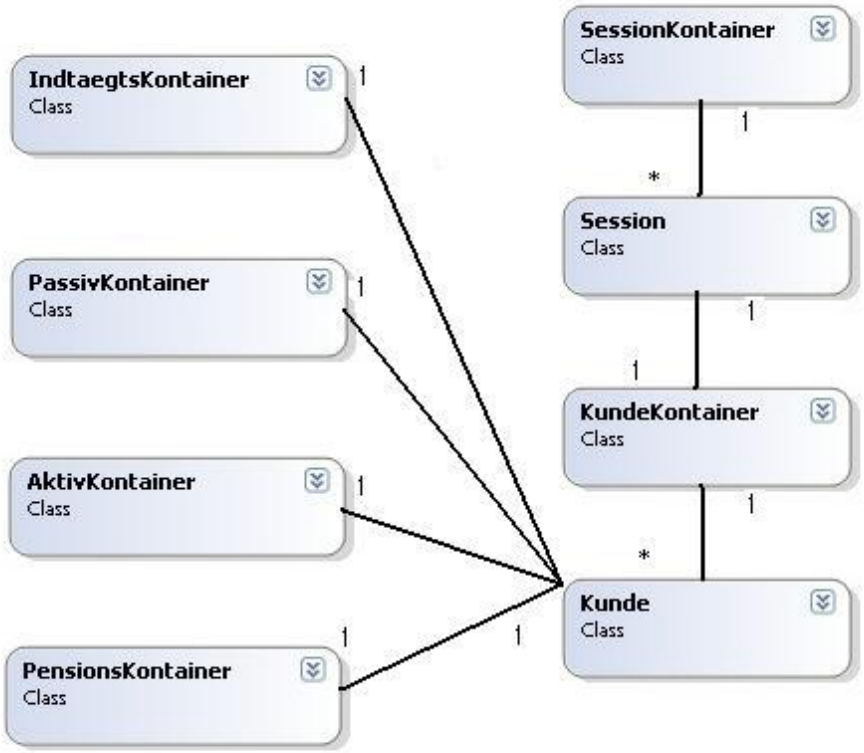
Gennem af entitetsklasserne vil blive gennemgået oppe fra og ned. Øverst oppe har en kontainer som indeholder en mængde 'Sessioner'. En 'Session' har en mængde af 'kunder' som opbevaret i klassen 'KundeKontainer'. En 'Kunde' har tilknyttet sig fire kontainere, som hverisær består af 'Intaegter', 'Aktiver', 'Passiver' og 'Pensioner'. Ovenstående beskrivelse kan ses på figur 10.1 på side 103.

De fire næste klassediagrammer viser forholdet mellem en kontainer klasser og den klasse som den er kontainer for. De viser også hvorledes af vi har gjort brug af arv. Diagram 10.2 kan ses på side 103, 10.3 kan ses på side 104, 10.4 kan ses på side 104 og 10.5 kan ses på side 104.

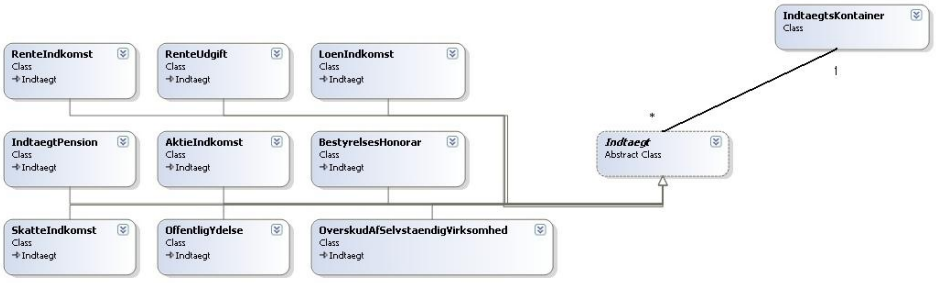
Efterfølgende var vi et diagram som viser at der en 'kontrol' klasse som hver styer mellem 1 og 3 forme/skræme. Dette repræsentere kontrol og view i design patternet MVC, som der kan læse om i afsnit 6.3.1 på side 76.

Det sidste klassediagram vi kan byde på indeholde forskellige klasser. Dette hendler om klassen der tager sig af at kommunikere be databasen, vores post nr. klasse og to klasser som kan bruges til at sortere med. Diagram 10.7 kan ses 106.

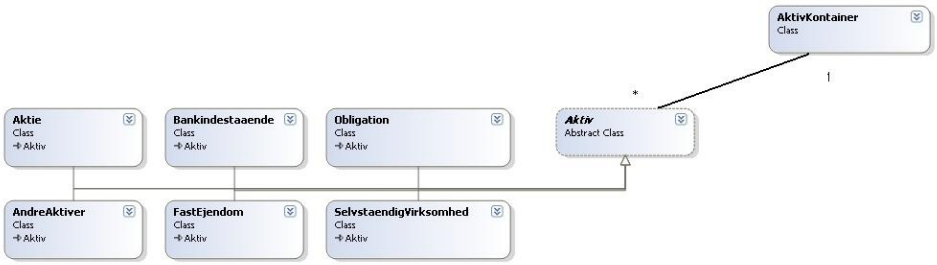
Til sidst vil vi vise de mange af samme diagram, men denne gang med detailer som medlemsvariabler og -metoder.



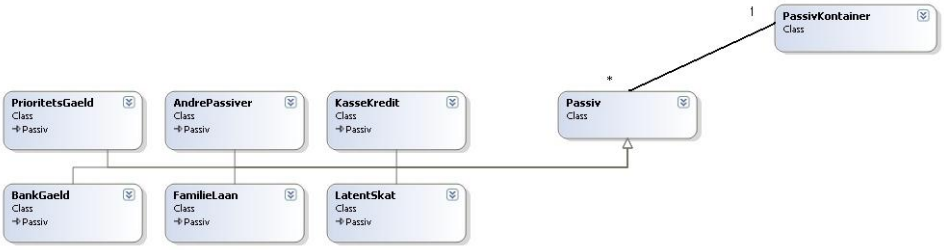
Figur 10.1: Klassediagram, Entiteter



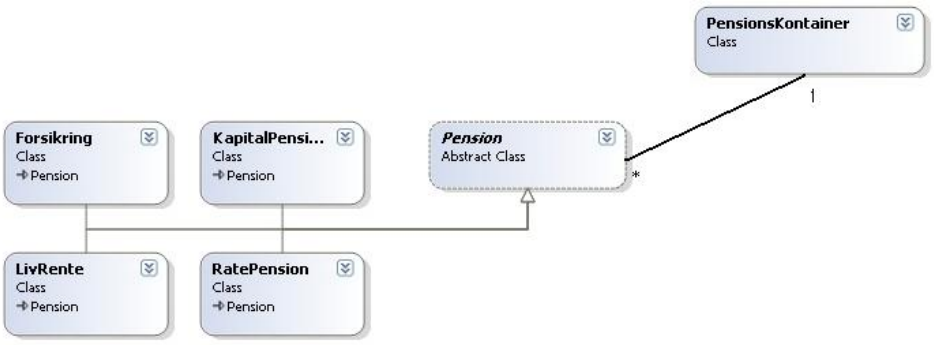
Figur 10.2: Klassediagram, Intægter



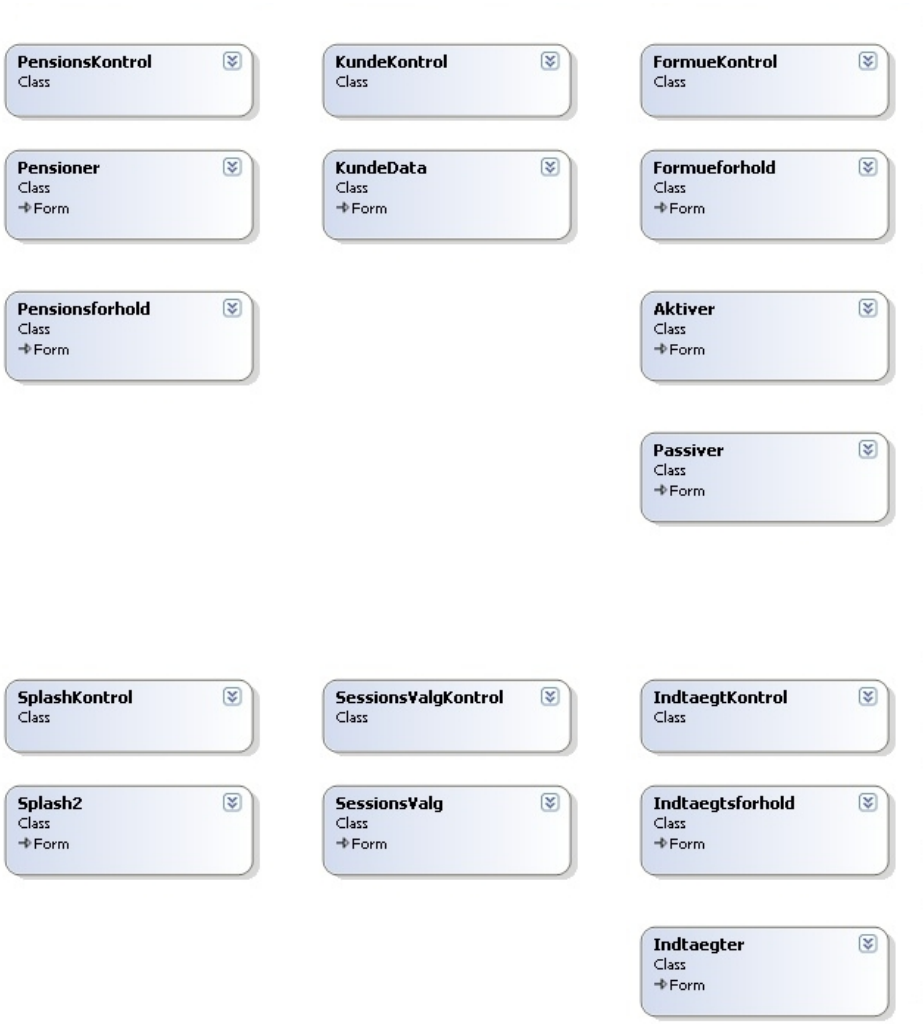
Figur 10.3: Klassediagram, Aktiver



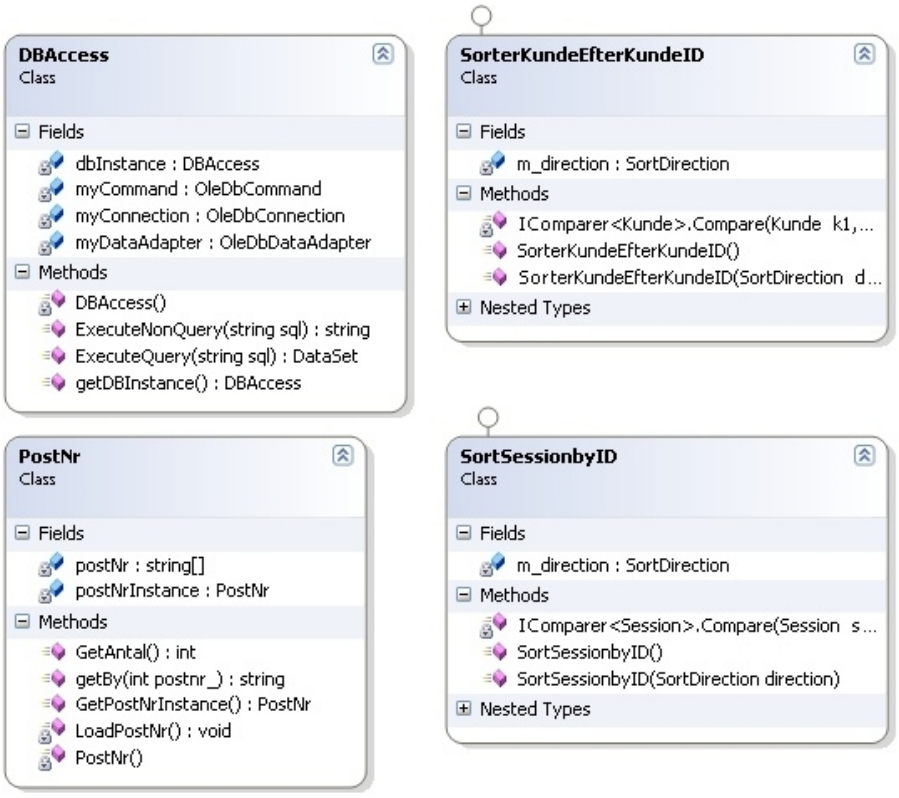
Figur 10.4: Klassediagram, Passiver



Figur 10.5: Klassediagram, Pensioner



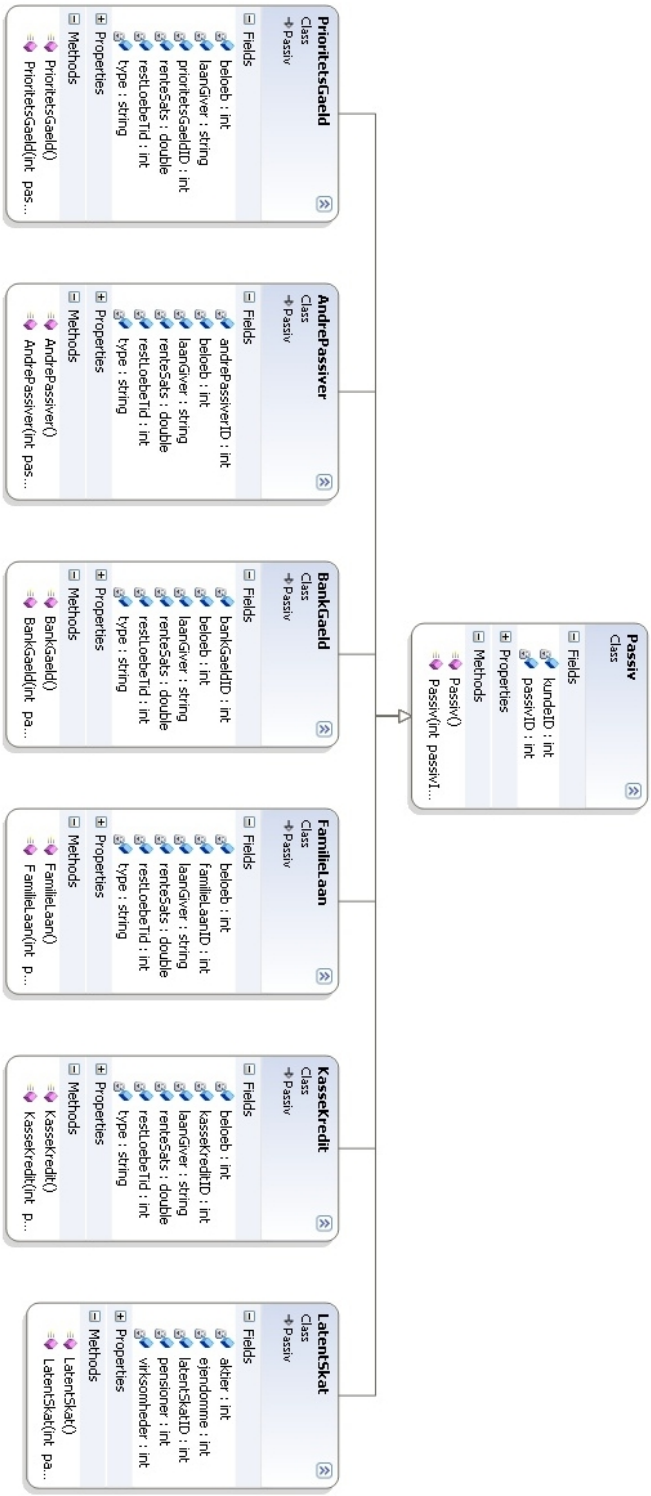
Figur 10.6: Klassediagram, Kontrol, View



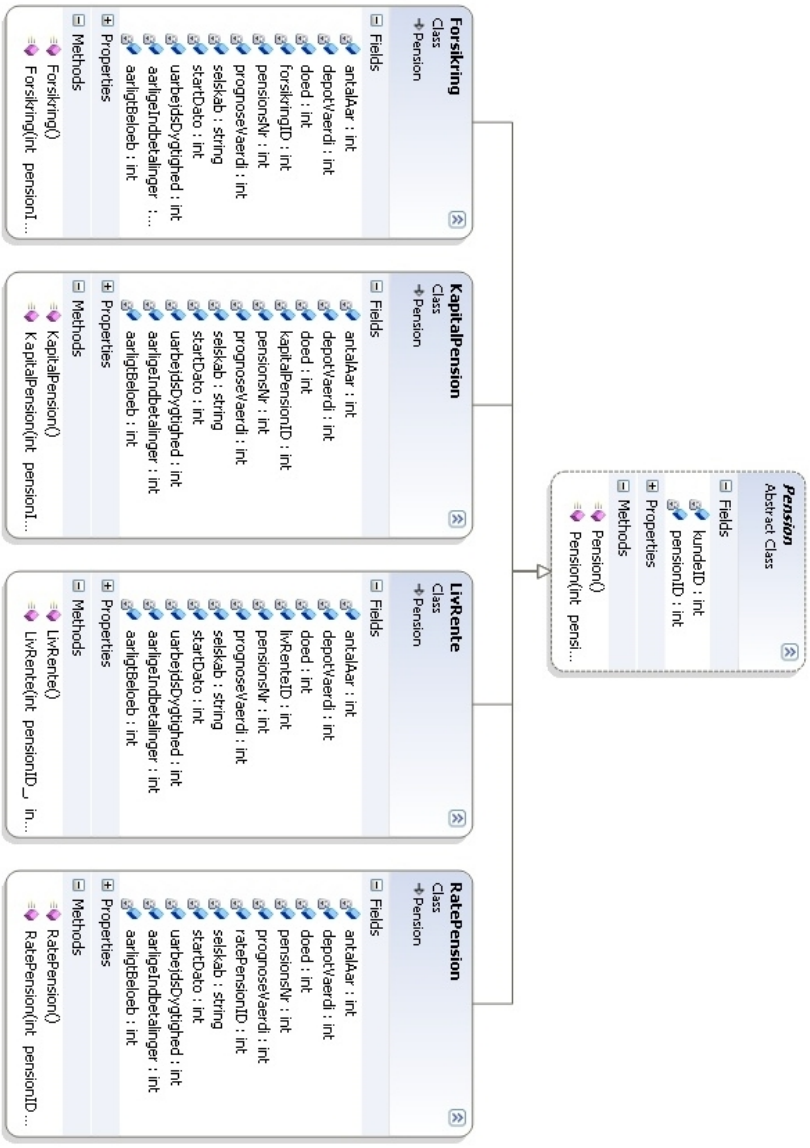
Figur 10.7: Klassediagram, Diverse

Figur 10.8: Klassediagram, Intaegt detailer

Figur 10.9: Klassediagram, Aktiv detailer



Figur 10.10: Klassediagram, Passiv detailer



Figur 10.11: Klassediagram, Pension detailer

10.1 EIK Banks oprindelige oplæg

PRIVATE BANKING VÆRKTØJ

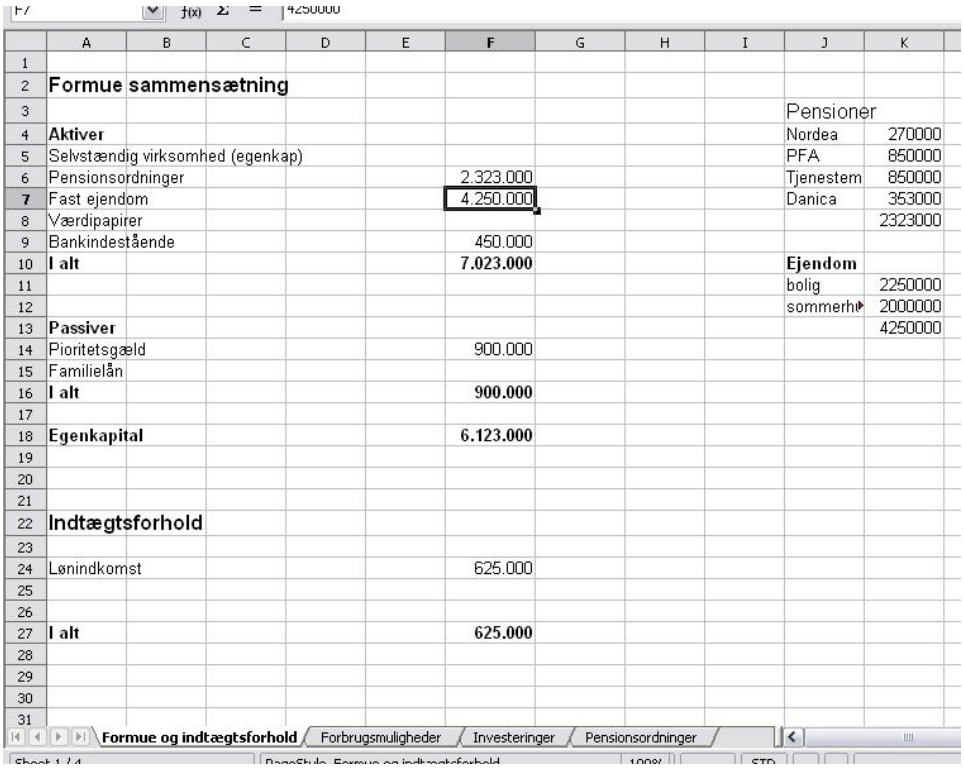
Indledning: Private banking består i rådgivning overfor formuende kunder indenfor investeringspleje, pensions- og skatteforhold, f.eks. i forbindelse med til- og fraflytning udlandet.
Grundlag: Forudsætningen for en professionel rådgivning af kunder indenfor Private Banking er et overblik over kundens - og ægtefælles indtægtforhold, formueforhold, arbejdssituation, pensionsopsparing, forsikringsdækninger samt fremtidige forventninger.
Dette overblik skabes i dag via registrering af alle ovennævnte oplysninger i Excel, hvorefter overblikket generes. Herefter skal man indtaste fornyede oplysninger for at simulere andre situationer i forbindelse med rådgivningen af kunden. Dette kan p.t. være en besværlig proces.

Fremtidigt værktøj: Projektgruppen opgave vil være at finde frem til en mere automatiseret løsning, som kan erstatte det nuværende excel ark.
Vi kan forestille os følgende løsningsmulighed:
En database, hvor kundens oplysningen registreres i forskellige tabeller.
En række andre tabeller/databaser med nyttige oplysninger, f.eks.

- offentlige pensioner
- test vedr. risikoanalyse vedr. risikoprofil
- pensions- og dækningsforhold for de foretrukne samarbejdspartnere/-forsikringsselskaber
- værdipapirer over foretrukne porteføljer, f.eks. Eik Banks modelportefølje
- skatteforhold i foretrukne udlande, f.eks. Frankring, Spanien, England m.fl.
- gængse realkreditlån, beregningsmotor, omlægning af lån

Noget af ovennævnte kunne tænkes at være links til andre websites.

Her ovenpå lægges et brugerinterface, hvor resultatet af kundens oplysninger vises i præsentabel format (mulighed for udskrift). Det skal endvidere være muligt at simulere forskellige muligheder/situationer under mødet med kunden, både med hensyn til indtægtsforhold, pension- og dækningsforhold, Investeringspleje samt skattemæssige forhold, dels på grundlag af kundens egne oplysninger og dels de faste oplysninger, jf. ovenfor. Kunden vil dermed få et relevant og grundigt overblik/grundlag for at træffe beslutninger.



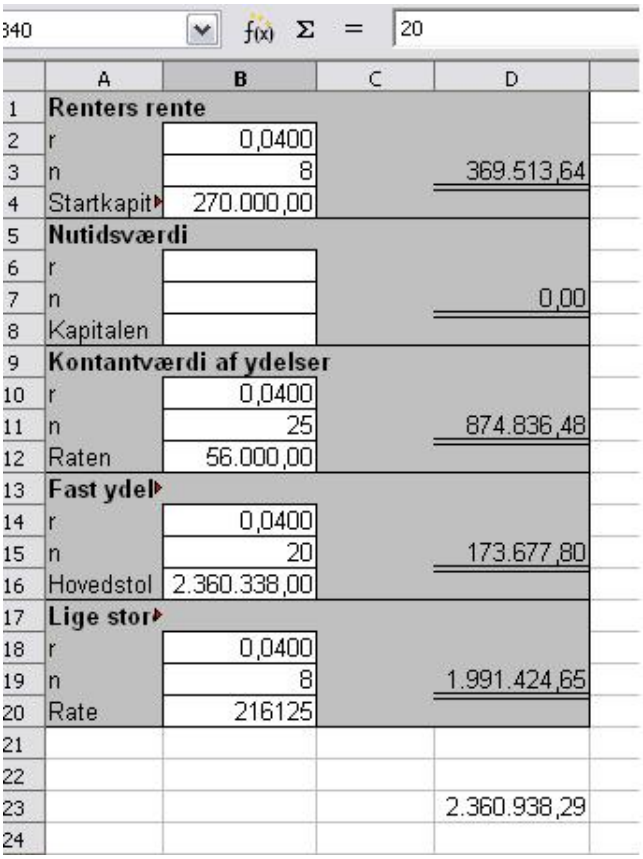
	A	B	C	D	E	F	G	H	I	J	K
1											
2	**Formue sammensætning**										
3										Pensioner	
4	**Aktiver**									Nordea	270000
5	Selvstændig virksomhed (egenkap)									PFA	850000
6	Pensionsordninger					2.323.000				Tjenestem	850000
7	Fast ejendom					4.250.000				Danica	353000
8	Værdipapirer										2323000
9	Bankindestående					450.000					
10	**I alt**					**7.023.000**				**Ejendom**	
11										bolig	2250000
12										sommerhu	2000000
13	**Passiver**										4250000
14	Pioritetsgæld					900.000					
15	Familielån										
16	**I alt**					**900.000**					
17											
18	**Egenkapital**					**6.123.000**					
19											
20											
21											
22	**Indtægtsforhold**										
23											
24	Lønindkomst					625.000					
25											
26											
27	**I alt**					**625.000**					
28											
29											
30											
31											

Formue og indtægtsforhold | Forbrugsmuligheder | Investeringer | Pensionsordninger

Figur 10.12: Formue og indtaegt

10.2 Det eksisterende system

Følgende udklip er fra et af de excelark banken bruger og vises her som et eksempel på det nuværende system.



	A	B	C	D
1	**Renters rente**			
2	r	0,0400		
3	n	8		369.513,64
4	Startkapit	270.000,00		
5	**Nutidsværdi**			
6	r			
7	n			0,00
8	Kapitalen			
9	**Kontantværdi af ydelser**			
10	r	0,0400		
11	n	25		874.836,48
12	Raten	56.000,00		
13	**Fast ydel**			
14	r	0,0400		
15	n	20		173.677,80
16	Hovedstol	2.360.338,00		
17	**Lige stor**			
18	r	0,0400		
19	n	8		1.991.424,65
20	Rate	216125		
21				
22				
23				2.360.938,29
24				

Figur 10.13: Forbrugsmuligheder

10.3 Morgenmøder

Morgenmøde 24-01-06

- Karakteristik af EIK: Ikke EDB-kyndige medarbejdere, brugere på normalt nivau.
- Valg af database? Access indtil videre. Da der ikke bliver meget pres på med 4 medarbejdere. Det ligger også i kortetene at vi bør gå efter en kostfri løsning. Afprøv en simpel database på EIK filserveren.
- Webservice? Kan filserveren bruges til webservices
- Rapportstruktur: Kig gamle rapporter igennem.
- Latex rapport header: Lav en header til texfilerne til rapporten.
- Videoproto af private banking session: Få Dion til at vise hvordan et realistisk kunderforløb ser ud. Se nuværende arbejdsgange og få ham til at sætte ord på de nuværende problemer.

24									
25									
26	**Renters rente**				**Renters rente**				
27	r	0,0400			r	0,0400			
28	n	8		483.104,87	n	8		483.104,87	
29	Startkapit	353.000,00			Startkapital	353.000,00			
30	**Nutidsværdi**				**Nutidsværdi**				
31	r				r				
32	n			0,00	n			0,00	
33	Kapitalen				Kapitalen				
34	**Kontantværdi af ydelser**				**Kontantværdi af ydelser**				
35	r				r	0,0400			
36	n			#VALUE!	n	10		437.988,37	
37	Raten				Raten	54.000,00			
38	**Fast ydel**				**Fast ydelse på lån**				
39	r	0,0400			r	0,0400			
40	n	20		89.787,54	n	20		32.228,81	
41	Hovedstol	1.220.242,00			Hovedstol	438.000,00			
42	**Lige stor**				**Lige store ydelser vokser til**				
43	r	0,0400			r	0,0400			
44	n	8		737.138,10	n	8		737.138,10	
45	Rate	80000			Rate	80000			
46									
47									
48				1.220.242,98				1.220.242,98	
49									
50									

Figur 10.14: Forbrugsmuligheder 2

	A	B	C	D	E	F	G	H	I	J
1	**Pensior**									
2										
3	**Årlige indtægter:**			**fra 2013**	**fra 2018**	**fra 2023**	**fra 2013**	**fra 2018**	**fra 2033**	
4				10 år	10 år	10 år	20 år	20 år	20 år	
5	Danica			150.000	150000		90000	90000		
6	Nordea			291.000	291000		173000	173000		
7	PFA			54.000	54000		32000	32000		
8	Tjenestemandspens.			56.000	56000	56000	56000	56000	56000	
9	ATP				20000	20000		20000	20000	
10	Folkepension				57000	57000		57000	57000	
11	**I alt**			**551.000**	**628000**	**133000**	**351000**	**428000**	**133000**	
12										
13										
14										
15	**Engangsbeløb**		**før afg.**	**efter afg.**						
16										
17	PFA		625358	375.215						
18										
19										

Figur 10.15: Pension 1

20					
21	**Indvalidedækning**				
22					
23	**Årlige indtægter indtil pensionering**				
24					
25	PFA		79.274		
26	Danica		217.872		
27	**I alt**		**297.146**		
28					
29	**Engangsbeløb**				
30					
31					
32	**I alt**		**0**		
33					
34					
35	**Dødsdækning**				
36					
37	**Årlige beløb**				
38					
39	PFA		96.577		i 10 år
40	Danica		26.614		i 10 år
41	**I alt**		**123.191**		
42					
43					
44					
45					
46	**Engangsbeløb**		**Brutto**		**Netto**
47					
48	Danica		388.000		232.800
49					

Figur 10.16: Pension 2

- Lave en tidsoversigt: Lave en tom 'skal' over tidsforløbet.
- Login delen: Påbegyndelse af logindelen er en mulighed allerede nu.
- Kigge på mulige faldgrupper i forløbet som det ser ud nu (mild risikoanalyse).
- Fastlægge filosofierne:
  - Crack the hardest nut first.
  - 'Stille dumme spørgsmål filosofien' Ingen spørgsmål er for dumme. Modvirker kommunikationsvanskeligheder.
  - KISS-Keep It Simple Stupid / KISBI: Keep It Simple But Intelligent
- Fastlægge arbejdsformerne: Spiralmodellen. Daglige møder efter Scrumforbillede. Der er 7 iterationer. 1 uge er planlægning og installation. Sidste 2 uger er til rapportskrivning. Køre efter fastlagte ugentlige mål. Det kan være et delsystem eller en papirmodel. el. Målet skal være færdigt når vi går hjem torsdag. Afholde et ugentligt informationsmøde med EIK kontakterne. Primært for at vise fremskridtet, engagement og for at hindre at vi kommer for langt ud på et sidespor. Dette ugentlige møde afholdes fredag formiddag (11-11.30 tiden). NOTE: Vi skal informere EIK medarbejdere på forhånd om hvor meget tid vi regner med at der går ved samtaler og andre sessioner.

Morgenmøde 26-01-06 Deltagere: Klaus, Jes og Christian

- Ting der skal ske i dag 26-04-06
  - 1.Afklaret om hvorvidt vi kan gøre bruge en database server.
  - 1.Vi skal have Johann til at tjekke med SDC hvilke ting vi kan få lov til på deres file-server, og hvilket OS den køre.
  - 2.Hvis der er Win2003Server om vi så kan bruge IIS (som enten er instelleret eller som vi må installerer) (dette skal undersøges).
  - 3.Der er møde mellem EIK og SDC på mandag.
  - 4.Hvis ikke vi kan bruge en database server, så bruger vi bruge en access database på et delt netværks-drev.
  - 2.Afklaret om hvorvidt vi kan lave en web-server på deres file-server. Ene og alene for at give os selv så mange udviklings muligheder som muligheder.
  - 3.Noterne fra mødet i går med Dion (video session) skal skreves sammen, for at fjerne dublikater.

  - 4.Kort snakke om det møde med Dion for at sikre at vi har samme opfattelse.
  - 5.Lave en skabelon/disposition til fredags mødet med EIK.

- Ting som tidligst skal laves fra om fradagen d. 27-01-06 På baggrund af video-session, Detaljeinformationsbehov (liste fra Dion 24-01-06) og regneark kan vi udfærdige prototyper.

Morgenmøde 27-01-06 Deltager Klaus, Jes og Christian.
Dagsorden.

- Der skal bestilles papir artiker.
- Vi skal have planlagt næste uge.
- Vi skal have Kim til at prioritere deres krav til systemet, samt at klarlægge vigtigheden af pensionsdelen.
- Opsummering af fredagsbriefing med EIK.
- Forberedelse til LoFi Prototyping session:
  - 830-930 Gennemgå materiale (Excel ark, printede regneark)
  - 930-1300 Kreativt stadie. Formål: En masse kreative tanke. OVER-ORDNET!
  - 930-1030 Individuelt udarbejde en løsning i LoFi.
  - 1030-1145 Vurdering i plenum. Diskuter for og imod.
  - 1145-1230 Frokost
  - 1230-1300 Beslut på baggrund af diskussionen hvad der er den bedste løsning.
  - 1300-1430 Konstruer prototypen.
  - 1430-1600 Uddeleger roller, og planlæg sessionen (foretages tirsdag kl 9.30)
  - 1600- Afslut dagen. Opsumer i dagbogen.

Morgenmøde 01-02-2006

- Der skal sammenlignes indtryk og noter fra de 2 LOFI prototype-sessioner vi har afholdt og de indsamlede data skal analyseres og sammenfattes.
- Problemformuleringen er oprettet som dokument under 'Rapport' og skal skrives færdig.

- Møde med Dion er booket til i morgen torsdag (02-02-06) hvor vi skal have et crash-kursus i pensionsteori kl 14.
- Planlægningen af fredags-briefing påbegyndes.
- Vi skal kigge nærmere på at specificere krav og afgrænsninger til og af systemet. Det samme gør sig gældende med at 'knække den hårdeste nød først'. Dette skal vi komme nærmere ind på, vi skal identificere den håreste nød.
- Det er ved at være på tide at vi kommer i gang med rapportskrivning. Her tænkes specielt på virkomshedsprofil, indledning og problemformulering.
- På fredag skal næste uge planlægges.
- Når fredags-briefing er overstået skal vi have en snak med Pia om projektet, produktet og systemudviklingen.
- Vi skal huske at drikke nogle bajere efter arbejdstid på fredag, vigtigt at få tankerne lidt væk.
- Tilføjelse: Ide: For at indsamle krav til præsentationsdelen af applikationen kunne der afholdes en brainstorming session/møde med Dion og Kim. Vi skal skære ind til benet og få de vigtige ting på banen. Hvad skal en præsentation for kunden indeholde. Hvilke grafer og på hvilke tal skal graferne laves. Vi skal arbejde os hen imod at få specificeret de enkelte indtastningsfelter i applikationen.

Morgenmøde 03-02-06 Deltagere Klaus, Jes og Christian

Planlæge næste ugens forløb. Opsamling på samtalen med Pia. Opsamling på fredags brifing

Morgenmøde 06-02-06 Plan for i dag

Vi skal undersøge om det er planen at der bliver rene indtastnings opgaver for sekretærerne. Der skal laves nye og pæne items til den næste prototype. Dette går Jes og Christian i gang med. Vi skal have diskuteret vore mappe struktur og placeringen af vores filer. Der er en reel risiko for at miste overbliket over alle vores noter. Vi skal kigge nærmere på at identificere den hårdeste nød - dette skal vi komme nærmere ind på.

Morgenmøde 07-02-06 Plan for i dag

Vi skal have diskuteret vore mappe struktur og placeringen af vores filer. Der er en reel risiko for at miste overblikket over alle vores noter. Den nye prototype skal laves færdig, så den er klar til brug ved session. Principper og problemformulering skal behandles. Vi skal have booket prototype-møde, vi afsætter en time til sessionen. Alt efter hvor meget tid vi har til overs kan vi kigge nærmere på et studieområde og endvidere få skrevet noget ned om prototyping.

Morgenmøde 08-02-06 Plan for i dag

Kigge nærmere på et studieområde. Endvidere få skrevet noget ned om prototyping. Problemformuleringen skal vi ordentlig i gang med. Kigge på Latex. Ang skrive afsnit sammen (Bibtex). Misc.

Morgenmøde 08-02-06 Deltagere: Klaus, Jes og Christian

Alle noter skal skrives ind. Vi skal have skrevt vores tanke ind i prototyping dokumenter, omkring det netop afholde session. Vi skal have lagt lyd og billed fra session skal lægges op til deling. Vi skal have indkaldt til fredagsbrefing.

Morgenmøde 10-02-06 Dagsorden

Planlægge og afholde fredagsbriefing. Sammenskrivning af noter fra session 3. Rapport-skrivning: Problemformulering, Oprydning i mappe-strukturen. Snakke om MVC og arkitektur. Planlægge uge 7. Undersøge muligheder for litteratur til studieområdet. Indledende tanker omkring næste uges designfase.

Morgenmøde 13-02-06 Dagsorden

Tage hul på design. Identificering af klasser Strukturdokument (navnekonventioner) Snakke om MVC og arkitektur. Undersøge muligheder for litteratur til studieområdet. Database tilgang undersøges. Test MVC-templaten af. Skrive om MVC.

Morgenmøde 14-02-06 Dagsorden

Database tilgang undersøges: Kompilere dbTest og smide det på EIKserveren for at teste det. Tage hul på design. Identificering af klasser Struktur-

dokument (navnekonventioner) Snakke om MVC og arkitektur. Undersøge muligheder for litteratur til studieområdet. Skrive om MVC.

Morgenmøde 15-02-06 Dagsorden

Såfremt at Johann er rask og på arbejde, skal vi teste db tilgang mellem hans maskine og filserveren. Design: Objekt-identifikation → pre-state klassediagram.

Navne-konventionering.

Morgenmøde 16-02-06 Dagsorden

Såfremt at Johann er rask og på arbejde, skal vi teste db tilgang mellem hans maskine og filserveren. Mapping til DB -¿ DB-diagram. Arv i forbindelse med C# og DB. Navnekonventionering. Oprette et C# projekt i Visual og samle de forberedte entitetsklasser. Vi skal have fat i Kim angående spørgsmål i forbindelse med Pensionsforhold (popup), samt Obligationer. Forberede fredags-briefing.

Morgenmøde 17-02-06 Dagsorden

Forberede og afholde fredagsbriefing. Hvis Johann er blevet rask → test DB. Fredag er primært en skrive-dag, hvilket betyder at vi skal arbejde med rapporten. Problemstillingen/er, en mere specifik beskrivelse af dette område. Design patterns, observer mønstret og vores formål med at bruge design patterns. Uddybning af de resterende områder. Principperne for mapping af DB og vores valg af princip. Planlægning af næste uge (uge 8).

Morgenmøde 20-02-06 Dagsorden

Der skal kigges på/kodes kontainer-klasser til entitetsklasserne, og kunde & session skal kodes. Databaseproblematik; performance (hvornår/hvor ofte skal der læses fra db'en, hvor skal denne tilgang ligge?)

Morgenmøde 21-02-06 Dagsorden

Navnekonvention: Alle reserverede SQL ord i query's skrives med stort. Kode GUI: dynamisk indhold på Aktiver, Passiver, Indtægter og Pensioner. Generel kodning af systemetet Hvordan får håndterer vi lister (dataudtræk, søgning og sortering) Session & kunde

Morgenmøde 22-02-06 Dagsorden

Jes skal opdateres. De resterende kontrol-klasser skal laves. De resterende entitets-klasser skal kobles sammen med Kunde. Database-klassen skal laves som en singleton. Hertil skal der skrives i rapporten, samt klargøres et eksempel. Hver kunde skal kunne loade sine Indtaegter, Aktiver, Passiver og Pensioner.

Morgenmøde 23-02-06 Dagsorden

Kode Lav 'gem-i-database' strukturen. Videre kodning af 'hent-fra-database' strukturen, som blev lavet i går. Bestemme hvordan programmet skal starte, hvad skal der ske når programmet starter, 'velkomstskærm' → lave en opstartsform.

Lavet og udsendt menu til fredagsbriefing.

Morgenmøde 24-02-06 Dagsorden

Afholde fredags-briefing. Opdatering af klassediagram i forhold til kode og opdatering af kode (navngivning). Oprydning i dokumenter/filer. Skrivedag: Database (normalisering, hvordan der skal gemmes etc.) Indledende til projektet (problemstilling etc.) Bedre/mere specifik beskrivelse af krav. Bedre/mere specifik beskrivelse af afgrænsninger.

Tale mere om afgræsning af projektet.

Morgenmøde 27-02-06 Dagsorden

Notesystemet: Afgrænsning / en mulighed for at 'shine' systemet lidt op. Hvordan ser startsiden til systemet ud? Man skal vælge en tidl session eller starte en ny. Kodning: Load/save funktionerne skal færdiggøres. Skal denne funktion optimeres? Færdiggørelse af et skærmbillede og bruge det som skabelon til resten af systemet. Også med henblik på fremvisning af system på fredagens briefing. GUI. Begynde med velkomstskærmen. Kæde teorien sammen med systemet. Forberede noget til Pias besøg.

Morgenmøde 28-02-06 Dagsorden

Kodning Opstarts screen skal forbedres. Navigeringen skal færdiggøres. Overordnet design Skal der sendes hele 'sessions' objekter mellem alle klasserne? Pia kommer idag. Vi skal snakke om hvad der specifikt skal stå i rapporten. Opbygningen / struktur af rapporten, hvor meget skal det ene og det andet emne fylde (f.eks Fokusområdet GUI og til dels Design Patterns). Vi skal snakke afgrænsning af systemet og dets funktionalitet.

Morgenmøde 28-02-06 Dagsorden

Arbejde videre med fremsending af Sessionsobjekt. Note skal læses ind i kundedata-formen. Kod videre.

Morgenmøde 02-03-06 Dagsorden

Indbydelse til fredags-briefing skal laves og fremsendes. SaveKunde funktionen skal tjekkes igennem, da der er mistanke om fejl i denne funktion, der skal formentlig sendes et sessionsid med over. Der skal arbejdes med 'Indtaegtforhold' og dens popup 'Indtaegter'. Vi tager udgangspunkt i én af indtaegtforholdende og koder den igennem.

Morgenmøde 03-03-06 Dagsorden

Afholde fredagsbriefing: demo af applikationen. Skrive-dag: Klaus har noget til DB Polymorfi Pias noter/forslag/input/meninger/idéer/

Prøve at booke et møde med Kim og Dion for at få kvantificeret krav.

Kode Der skal arbejdes med gem-algoritmen

Morgenmøde 06-03-06 Dagsorden

Kvantificering af krav med Kim og Dion. Forberede møde. Book et møde til tirsdag eftermiddag. CRUD (create/read/update/delete) funktionalitet. Research GUI / print. Kode Print funktionalitet. GUI funktionalitet. De 10 gode råd. Treeview. inkl. fejlcheck. Lav en algoritme.

Kommende dage: Skrive noget om polymorfi.

Morgenmøde 07-03-06 Dagsorden

Kvantificering af krav med Kim og Dion kl 14. Forberede møde. Bruge kravslisten. Smid CRUD på. Research GUI / print. Kode Kigge på Factory Design Patterns. Print funktionalitet. Lave en eksperten. GUI funktionalitet. De 10 gode råd. Treeview. inkl. fejlcheck. Lav en algoritme.

Evt / Kommende dage: Skrive noget om polymorfi.

Morgenmøde 08-03-06 Dagsorden

Noter til møde med Kim, Claus og Dion. Research GUI - finde Bogen / print. Kigge på Factory Design Patterns. Print funktionalitet. Check op på Eksperten spørgsmål, hvordan portes objekt med serialisation til XML og tilbage igen. GUI funktionalitet. De 10 gode råd. Treeview. Update mangler. Hvordan for vi sat beløb på på oversigt under Indtægtsforhold. Vi skal have

lavet en kravliste. Skal have dækket alle funktioner i programmet ind med krav. Have lavet et udkast hvordan rapporten fra udprintdelen af systemet kunne se ud.

Evt / Kommende dage: Skrive noget om polymorfi.

Morgenmøde 09-03-06 Dagsorden

Videre med Crystal Reports og XML serialization. Knap til at tilføje en post på 'Indtaegter' der ikke lukker vinduet. Treeview problemet skal løses. Implementation af reglerne for hvornår elementer på brugergrænsefladen skal være synlige/brugbare (impl. de 10 regler i vores GUI). GUI litteratur. Kigge på 'abstract factory' design patterns. Forberede og udsende invitation til fredagsbriefing.

Morgenmøde 10-03-06 Dagsorden

Fredagsbriefing. Skrivedag. Skrive om GUI / HCI. Polymorfi / arv. Opdatere design Patterns generelt, se vores gamle Design Patterns aflevering Checke op på dokumentet fra Pia mødet. Rydde op i vores dokumentstruktur. Arbejde videre med Crystal Reports.

Morgenmøde 14-03-06 Dagsorden

Videre med Crystal Reports og dertilhørende dataset. Vi skal have lagt rediger og slet på Indtægtsformen. GUI regler skal implementeres. GUI design. Ændre fra rådgiver til kundenavn på 'tidl session'.

Morgenmøde 14-03-06 Dagsorden

Videre med Crystal Reports og dertilhørende dataset. Rediger-funktionalitet på Indtægtsforhold. GUI regler skal implementeres. GUI design. Kommentering af koden.

Morgenmøde 15-03-06 Dagsorden

Videre med Crystal Reports. Hvis ikke at vi kan få det til at virke som vi har foreslået at det skal virke, dvs. med flere koloner, så kunne vi lave én rapport pr kunde. //Vi kan/skal undersøge hvordan man sætterdynamiske felter på rapporten. Vi er blevet enige om at vi laver en rapport pr kunde. GUI regler skal implementeres. Vi mangler indtægter, login og sessionvælger. GUI design. Vi skal finde ikoner til vores knapper. Og vi skal efterleve gestalt lovene. Kommentering af koden. Der mangler kommentarer til indtaegt og til en af dens subklasser. Vi har haft en diskution om hvorvidt alt kodning

skal lukkes på fredag d. 17-03-06 eller om GUI design skal være åben for forandringer i forbindelse med at vi bliver klogere på det område. Klaus mener at alt kode skal være lukket og at vi i rapport må skrive hvad vi kunne have gjort bedre med henhold til GUI. Jes og Christian vil gerne holde GUI åben så ændringerne kan blive lavet sideløbenede med at vi bliver klogere. Vi diskuterer dette igen efter frokost.

Morgenmøde 16-03-06 Dagsorden

Crystal Reports: Vi er blevet enige om at vi laver en rapport pr kunde. Primært mål: Vi skal have noget simpelt data ud - vi forventer at ende med en skrabet version af rapporten. Vi skal have hørt vejleder Pia om hvordan man bør forholde sig til et krav fra en virksomhed, som ikke kan løses/ikke kan løses fuldstændigt. Hvordan bør man forholde sig i forhold til skolen (NB) og i forhold til virksomheden (EIK). GUI regler skal implementeres. Vi mangler sessionvælger. (slet og rediger i Indtægtsforhold bliver ved med at være synlige, selv når der ingen indtægt er). GUI design. Vi skal efterleve gestalt lovene. (beskrive disse gestalt love). Skal implementeres også. Kommentering af koden. Der er skrevet kommentarer til det meste kode, de skal dog generelt gås igennem og rettes til hvor nødvendigt. Der mangler bla parameterlister og beskrivelser af disse, samt beskrivelser af hvad non-void funktionerne returnerer. Vi skal indkalde til Fredags Briefing. Her skal der tages stilling til om vi vil holde flere fredagsbriefings. Dette skal kommunikeres ud til EIK på fredagsbriefingen. Forslag: At udlevere hvad vi har skrevet af rapport indtil videre og lade dem læse det igennem. Vi kunne evt. afholde møde mandag hvor vi kunne tage imod evt. kritik. Vi bør nok også afholde en lille session hvor tester programmet af med de kommende brugere. Vi bør måle fremgangen i forhold til tidl. arbejdsgange, og lade dem teste programmet lidt af.

Morgenmøde 17-03-06 Dagsorden

Crystal Reports: Formatering af rapporten. Venstrestilles. Fredags Briefing skal afholdes. Skrive noget om prototyping. Rydde op i mappestrukturen. Opstarte med at skrive i Latex. Se hvor mange sider vi har allerede? GUI regler. TABs. (slet og rediger i Indtægtsforhold bliver ved med at være synlige, selv når der ingen indtægt er). GUI Design gestalt lovene. Kommentering skal der arbejdes lidt mere på. Skal nok gåes lidt igennem. Mangler måske parametre nogen steder. (lav prioritet, evt først efter forløbet her er afsluttet).

ANG: Fredags Briefing. Her skal der tages stilling til om vi vil holde flere fredags briefings. Dette skal kommunikeres ud til EIK på Fredags Briefingen. Forslag: At udlevere hvad vi har skrevet af rapport i den sidste del af ugen

og lade dem læse det igennem. Vi kunne evt. afholde møde mandag hvor vi kunne tage imod evt. kritik. Vi bør nok også afholde en lille session hvor tester programmet af med de kommende brugere. Vi bør måle fremgangen i forhold til tidl. arbejdsgange, og lade dem teste programmet lidt af.

Morgenmøde 20-03-06 Dagsorden

Det sidste afsnit i prototyping skal gennemgås. Dermed bør Prototyping være tæt på at være færdigt. Samle op på 1. hånds noter og vores dokumenter generelt. Finde ud af hvad vi skal bruge som rød tråd (gennemgangs-eksempel) Virksomhedsprofil, indledning, problemstilling- og formulering. Principper: Hvordan er det så gået med at bruge disse principper? Studieområde dokumentet. Database dokumentet. Test: hvad skal bruges som testmateriale: Databasetilgang, gem-algoritmen.

Fra i dag, det vil sige nu, skriver vi direkte i latex-dokumenterne.

Morgenmøde 21-03-06 Dagsorden

Finde ud af hvad vi skal bruge som rød tråd (gennemgangs-eksempel). Arrangeret test af applikationen med brugere. Studieområde dokumentet. (GUI & Design patterns). Principper: Hvordan er det så gået med at bruge vores principper? Generel gennemlæsning af rapporten, og rette fejl i den. Test: hvad skal bruges som testmateriale: Databasetilgang, gem-algoritmen.

Morgenmøde 22-03-06 Dagsorden

Der skal laves et gennemgangseksempel (rød tråd). Test-sessionen skal forberedes. Generel rapport-skrivning til de forskellige afsnit. Evaluering af vores brug af de principper vi har defineret. Studieområder. Systemudviklingsmetode. Test, her kan vi beskrive hvad vi har gjort indtil videre.

De sidste rettelser fra i går skal tilføjes.

Morgenmøde 23-03-06 Dagsorden

Test-sessionen skal forberedes. Problemformuleringen skal der styr på. Konklusion opstartes, og rettes hen i retning af problemformuleringen. Crystal reports / udprintsfunktionalitet skal der skrives om. Design af Brugergrænseflade. Hvor skal vi hen med dette punkt? Generel rapport-skrivning til de forskellige afsnit.

Morgenmøde 24-03-06 Dagsorden

Test-sessionen skal forberedes. Test sessionen med deltager Kim skal afholdes. Vi skal have sendt rapporten ud til Pia og EIK medarbejderne Johann, Dion, Kim. Hvor ligger vores hovedproblemer: Hovedstudieområdet GUI, vi skal have skrevet en smøre til det emne. Vi skal have skrevet et dokument omkring udprintfunktionaliteten. Det volder også en del problemer at beskrive vores process, altså hvilke valg vi traf på hvilket tidspunkt, og hvorfor vi traf dette valg.

Morgenmøde 27-03-06 Dagsorden

Feedback på rapporten (rapporten har været smidt ud til Johann og Kim og Pia). Skrive. Beskrive vores metode ud fra undervisningsbogen. Altså bruge skabelonen fra SiP undervisningen til at beskrive vores metode. Bygge det op på denne måde: Beskrive hvordan den teoretisk er bygget op og derefter fortælle hvordan og hvornår den er blevet brugt. Vi skal have skrevet et dokument omkring udprintfunktionaliteten. Vores process, altså hvilke valg vi traf på hvilket tidspunkt, og hvorfor vi traf dette valg.

Morgenmøde 28-03-06 Dagsorden

Feedback på rapporten (rapporten har været smidt ud til Johann og Kim og Pia). Vi mangler Kim, Pia og lidt Johann. Skrive. Vi skal have skrevet et dokument omkring udprintfunktionaliteten. Nogle af dokumenterne har for lange afsnit. De skal laves læsevenlige ved at sætte afsnit ind. Afrundinger på de 'færdige' dokumenter. Prototyping. Manglende afsluttende evaluering. Testsessionen skal afsluttes. Rydde op i kravslisten. Mantra: Vores process, altså hvilke valg vi traf på hvilket tidspunkt, og hvorfor vi traf dette valg.

10.4 Dagbøger

I dag d. 23-01-06

- Skal have overblik over rapporten. Hvordan er vores forestilling om den overordnede struktur. Der skal være fyld til 100 sider (puha :). Vi skal i gang med at skrive på rapporten med det samme og skal absolut ikke vente til sidst med at gå i gang.
- Vi har snakket om at vi skal kunne tage udgangspunkt i de forskellige områder når vi diskuterer, eller rettere forsøge at adskille de forskellige områder: analyse, design, implementering. Fordelen er f.eks at vi ikke begrænser vores kreative tankegang af snævre kodetankegange i en analyse situation hvor kreative muligheder skal udforskes. Altså skal vi forsøge at være mere bevidste på hvilken situation vi befinder os

i og tage den rette kasket på. Altså ikke noget med at tage analyse-kasket OG implementeringskasket på. OFTE FORUDSÆTTER DE FORSKELLIGE SITUATIONER HINANDEN OG BØR DERFOR UDFØRES I GENSIDIGT SAMSPIL ;)

- Vi opfatter hver især de svar vi får forskelligt, hvilket kan ses som et resultat af dårlig møde-teknik og/eller som et resultat af et kommunikationsvanskeligheder (i gruppen og mellem EIK og os).
- Vores møde i dag var alt for uprofessionelt, vi skal foreberede os bedre (roller, ikke snakke i munden på hinanden, have spørgsmål klar før mødet, fælles agenda).
- Vi har allerede nu en klar forestilling om at vi skal have en database ind over vores løsning. Efter at have taget en lille samtale om fordele og ulemper med database eller ej.
- EIK har en filserver, som databasen kan køre på. Dette skal testes.
- Vi har også løst snakket om hvilke SU-metoder der skal indover udviklingsforløbet. Der er stemning for eksperimentel prototyping og brug af UML.

I dag d. 24-01-06

- Applikationsorienteret:
  Vi er blevet usikre på om en databaseret løsning overhovedet kan lade sig gøre, da den pågældende filserver findes ude på SDC.
- Vi har lavet en plan for ugen.
- Aftalt møde med Dion: Gennemgang af nuværende forløb med video for at fastholde.
- Der er blevet udarbejdet en tidslinie vi kan kigge på og overskue hvor langt vi er nået i forløbet.
- Der er blevet lavet en header i Latex til rapporten.
- Der er blevet udarbejdet en udviklingsmodel for projektforløbet.
- PopstGreSQL Test database installeret på vores PIII serveren i vores lokale.

I dag d. 25-01-06

- Vi har installeret postgresql på klaus's maskine og på serveren. Vi kan komme i kontakt med klaus's postgres, men ikke den på serveren.
- Vi mangler CD'en til windows 2003 server, som skal bruges til at installere IIS på serven. Denne CD skulle være klar til i morgen. Så kan vi teste om vi kan bruge web-services.
- Vi har lavet videooptagelser over arbejdsgangene hos Dion.

  Faldgrupper:

- Vi skal passe op ikke at mister fokus på den største risiko. Der kan være en tendens/risiko for at komme til at fokusere på en ting/emner der interessere os, frem for den 'hårdeste nød'.
- En løsning kunne være at lave en 'risiko analyse' for på den måde at kunne prioritere hvad der er vigtigst. Dette analyse dokument skal løbende holdes opdateret. Dokumentet skal ikke være super detaileret men heller ikke super simpelt.
- Misforståelser. Vi skal være obs på hvad og hvordan vi kommunikere internt og eksternt. En løsning kunne være: At vi skal arbejde på at få et fælles sprog, ved at stille mange spørgsmål, dumme spørgsmål og uddybende spørgsmål. Vi kan også få et fælles sprog, EIK og os selv imellem, gennem brug af eksperimentielle udviklingsmetoder. F.eks. Lo-Fi. Man kunne lave en fælles ordbog/ordliste, men dette bliver for administrativt tungt. Problemer med at få de reelle krav ud af kunden. Der er risiko for at forveksle kundens krav, altså de problemer som de tror de har, med de reelle problemer de har. Ikke sandsynligt at kunden har et fuldstændigt neutralt overblik. Der er også risiko for at vi ikke for dykket dybt nok ned i disse problemer og EIK folkenes arbejdesgange. Vi skal ikke stille os tilfredse med det første og bedste svar.
- En løsning kunne være: At bruge 5 times why?
- Manglende mulighed for brug af database server. Hvis ikke at vi kan bruge deres file-server som db-server må vi finde en anden løsning. En løsning kunne være at lave en af db-server på en lokal maskine. Enten en decideret server eller en access database.
- Vi har lavet et udrids til en indbydelse til den Ugentlige Briefing af EIK. Pia er inviteret til dette arrangement, som forgår på fredag.

I dag d. 26-01-06

- Video fra mødet med Dion i går er klippet sammen.
- Noterne fra videomødet er sammenskrevet.
- Win Server 2003 Enterprise edition medbragt. Serveren er installeret og der arbejdes i øjeblikket på forbindelse til PostgreSQL databasen gennem Visual Studio. IIS var ikke nødvendigt, men der skulle manuelt lukkes op for TCP/IP filteret.
- Spørgsmål til SDC skal afleveres til Johann. Der er desværre risiko for at vi først får endeligt svar mandag. Vi havde som ugentlig mål at skulle have afklaret databasespørgsmålet. Vi bliver nødt til, indtil videre i hvert fald, at gå ud fra at vi skal arbejde med en MS Access databaseløsning.
- Vi har tilmelding af Johann, Kim og Dion til vores fredagsbriefing, Klaus var forhindret. Pia har meldt fra, men vil gerne komme i næste uge. Vi nåede ikke at sende indbydelsen frem inden folk er gået hjem desværre. For fremtiden vil vi sende den ugentlige indbydelse ud torsdag kl 12.

I dag d. 27-01-06

- Vi har afholdt en glimrende fredags briefing. Taget noter til det.
- Bestilt materialer til LoFi prototyping i næste uge.

- Vi har planlagt lidt af det der skal ske i næste uge. Prototyping sessioner tirsdag (Kim) og onsdag morgen (Dion).
- Mandag er indtil videre afsat til at planlægge vores Lo-Fi prototype. Vi skal blandt andet lave et scenarie. Resultaterne behandles om eftermiddagen.
- Kim fortæller at vi skal betragte det som et stort fælles samspil. Det er ikke til at prioritere.
- Vi skal have alle data tastet ind vedrørende Formue / indtægter. Materialet til indtastning er altid på papirform.
- Altså indtastning af StamData. Efterfølgende sideopslag om udspecificeringer af f.eks pensionsordningner, aktier osv.
- henter den information det skal bruge fra det indtastede StamData. Kører nu sådan her:

1. Indtastning af stamdata.
2. Oversigtsbillede. (formue og indtægter)
3. Dyk ned i de underpunkterne hvor data er mere udspecificerede (pensioner, aktier osv.)
4. Ud fra disse udspecificeringer kan der så laves lidt scenarier: (havd sker der hvis du i morgen: blev invalid, ægtefælle døde osv.)

I dag d. 30-01-06

- Fulgte vores plan for udarbejdelsen af Lo-Fi prototype.

830-930	Gennemgå materiale (Excel ark, printede regneark)
930-1300	Kreativt stadie. Formål: En masse kreative tanker. OVERORDNET!
930-1030	Individuelt udarbejde en løsning i LoFi.
1030-1145	Vurdering i plenum. Diskuter for og imod.
1145-1230	Frokost.
1230-1300	Beslut på baggrund af diskussionen hvad der er den bedste løsning.
1300-1430	Konstruer prototypen.
1430-1600	Uddeleger roller, og planlæg sessionen (foretages tirsdag kl. 9.30)
1600-	Afslut dagen. Opsumer i dagbogen.

- Vi beslutter os til at gå i dybden, afgrænse os til 'indtastning' i vores prototype.
- Vi får kopieret materiale fra en eksisterende sag (Told og Skat, Bank, Pensionsoplysninger)
- Vi har lavet Lo-Fi prototyperne og skal have feedbacken på det i morgen på sessionen med Kim.

I dag d. 01-02-06

- Vi har afholdt en god prototype session med Dion. Masser af information.
- Vi har kogt meget ned på al den information vi har samlet sammen i prototype sessionerne med Dion og Kim og lavet et dokument over informationen.
- Vi har fået svar fra Pia, hun vil gerne komme forbi på fredag.
- Vi har fået begyndt på at skrive lidt på rapporten. Vi kan allerede nu tilføje til indledning, problemformulering.
- Vi har lavet en ny arbejdsgruppe 'Students' hvor vi har fået kontakt fra alle maskiner til en Access DB på vores server.
- Vi skal have forberedt lidt disposition af vores problemformulering til rapporten til fredagsbriefingen.

I dag d. 02-02-06

- Vi skal have sendt indbydelsen til fredags briefingen ud.
- Vi skal have et crash course i Pensioner med Dion kl 14-15. Målet hermed er at vi har en klar forståelse af grundprincipperne i pensioner i relation til applikationen.
- Vi fik foretaget et møde med Kim vedrørende krav til systemet og fik opklaret mange tvivlsspørgsmål, og fik nogle nye tvivlspørgsmål.
- Sammenfattet noterne fra de to første prototyping sessioner, og lagt Kims oveni.
- Arbejdet med krav generelt.

I dag d. 03-02-06

- Vi har afholdt fredags briefing
- Møde med Pia
- Opsamling på 'morgenmøde 01/02-06'.
- Vi har samlet op på fredagsbriefingen og mødet med Pia.
- Vi har udarbejdet plan for næste uge, hvori der indgår de løse ender som skal samles op indgår.

I dag d. 06-02-06

- Det er ved samtale med Kim blevet klargjort at indtil videre skal sekretærerne ikke stå for indtastning. Dette 'files under future upgrade possiblities'.
- Materialet til sessioner med Kim, Dion og Claus, hvor indtastningsfelterne skal klarlægges, er fabrikeret.
- Hårdeste nød problematikken er beskrevet for nuværende stadie i processen.

I dag d. 07-02-06

- Vi har fået vedtaget regler for brug af vores mappestruktur.
- Vi har vedtaget at opdele i '1.gangs' og '2.gangs' dokumenter. I 2. gangs dokumenterne samles konkret information om et bestemt emne.
- De bruges endvidere til at forhindre at vi 'drukner' i information, og skal altså give et overblik.

- Den formelle struktur ang. navngivningen af dokumenter i 1.gangsmappen er vedtaget.
- Vi har vedtaget ny måde at navngive dokumenter. Dato (år, måned, dag) + underscore + hint til emnet der behandles i dokumentet.
- Selve mappestrukturen på Tortoise repositoriet er opdateret.
- Fik nedfældet lidt principper
- Vi har næsten færdiggjort vores prototype.
- Aftalt og booket møde med Dion, Kim og Claus.
- Efterlyst kontorstol og telefon.

I dag d. 08-02-06

- Vi har fået skrevet en del på 2.håndsdokument om emnet prototyping.
- Vi har fastlagt vores studieområder til GUI, med design patterns som sekundær studieområde.
- Vi har påbegyndt research på studieområdet GUI. Der er skrevet lidt noter.
- En litteraturliste er blevet klargjort i latex.
- Vi har fastlagt et princip mere: Rotation af roller.

I dag d. 09-02-06

- I dag har vi udført en længerevarende session for at fastlægge de specifikke indtastningsfelter.
- Vi har skrevet vores noter ind til ovennævnte session.
- Der er tilføjet til rapporten: Studieområde, prototyping, database problematik.

I dag d. 10-02-06

- Vi afholdte en godt, men kort fredags briefing.
- Problemformuleringen er skrevet ind. Meget kortfattet og koncis.
- Mappestrukturen er ryddet lidt op, bla opretttet en skraldmappe.
- Der er fundet lidt litteratur om GUI. Gestalt principperne.

- Vi har skrevet noterne sammen om prototype session 3 (præcisering af indtastningsfelter)
- Vi har planlagt uge 7: (Vi skal træde ind på designdelen af systemet, vi skal teste DB connection)
- MVC skabelon testes af.

I dag d. 13-02-06

- Reasearchet på GUI. Skrevet 3 sider.
- Testet DB tilgang på vores egne maskiner med succes.
- Der er skrevet 2.hånds omkring design patterns.
- MVC templaten virker.

I dag d. 14-02-06

- Vi har arbejdet lidt på hvordan strukturen af klasserne kan bygges op. Der er skrevet et dokument, 'Identificering af klasser problematik' som ligger under noter i 1.hånd, hvor der står lidt omkring de problematikker vi stødte på.
- Vi er klar til at teste db forbindelse, men manglede Johann (mavesyg)
- Problemet med manglende dansk orddeling og manglende isbn er rettet i latex.
- Kørte død efter en hård middag og kaldte det en dag.

I dag d. 15-02-06

- Identificeret objekter og udformet et klassediagram.
- Gået i gang med at udforme et db-diagram. Her skal vi dog kigge mere på db-inheritance.
- Rettet lidt i GUI-teori-beskrivelsen.
- Snakket med Hugo omkring test af db, vi har aftalt at vi tager fat i Johann når han kommer i morgen.
- Spurgt Kim om de 4 poster i pensions-forhold virkelig skal være ens. Kim ville komme ned til os, men er formentlig blevet optaget af noget andet. Vi spørger Kim igen i morgen.
- Vi går hver især hjem og laver entitets-klasser for henholdsvis indtægtsforhold, aktiver og passiver. Pensionsforhold venter vi med da vi har uafklarede spørgsmål til dette punkt.

I dag d. 16-02-06

- Johann stadig syg, så ingen test af DB idag.
- Navnekonventionering er påbegyndt.
- Vi har snakket med Kim omkring fastlæggelse og præcisering af punkterne under Pensionsforhold. Ligeledes blev det opklaret hvilke punkter der skulle være under Formueforhold → aktiver → 'obligationer'.
- DBen er mappet og implementeret.
- De forskellige entiter (klasser) er blevet produceret siden i går aftes. De er nu samlet i et Visual Project, i hver sin mappe med dertilhørende namespaces.
- Fredags-briefingen er forberedt og menu sendt ud.

I dag d. 17-02-06

- Afholdt fredagsbriefing, hvor Kim og bankdirektør Brian Toft, Klaus og Christian deltog. Referatet fra dette møde ligger under Møder med EIK. Det er første gang Brian Toft er med, så han fik et indtryk af hvad det er vi er igang med at lave. Til mødet havde vi et udprint af klassediagrammet med, som Brian Toft og Kim fik at se.
- Johann kom tilbage og vi har nu omsider fået testet databasetilgangen fra Johanns maskine. Dermed kan vi gå ud fra at det ikke bliver noget problem at tilgå db'en fra EIKs workstations på deres filserver (netværksdrev k:) hos SDC.
- Kim var nysgerrig og ville gerne se hvad dette gik ud på, hvorfor vi har demonstreret vores test-applikation for ham. Om det levede op til hans forestilling om hvad en sådan test gik ud på, må stå hen i det uvisse ;-)
- Oprettet et dokument til beskrivelse af test og skrevet på dette.
- Skrevet omkring principperne for mapping af databasen og vores valg af model.
- Generelt snakket database og fundet ud af at primær-nøglerne i alle subklasser er unødvendige, hvormed vi sparer en kolonne pr. subklasse.
- Databasen er opdateret, unødige nøgler slettet i subklasser og pension re-mappet i henhold til 8d, resten af databasen forbliver i 8a.

I dag d. 20-02-06

- Kodet kontainerklasser for baseklasserne.
- Smidt ID's på samtlige entitetsklasser.
- Påbegyndt at lave kunde og kundedata klasserne.
- Tilføjet DBAccess klassen.
- Tilføjet et kald til baseklassen fra hver af subklassernes konstruktører.
- Tilføjet winforms (de involverede GUIs med indhold) (mangler indtastningsfelter)
- Kodet sessionsklassen.
- Skrevet om database tilgangsproblematik: performance. Hvornår og hvor ofte skal der læses og skrives, hvor skal det ligge i stukturen.

I dag d.21-02-06

- Tilføjet dynamisk indhold på Indtaegter, Aktiver, Passiver og Pensioner.
- Kontrollerklasse til 'Kunde' er tilføjet og er funktionel.
- Sorteringsklasser til sortering af Kunde- og Sessionslister er lavet.
- Sessioner og Kunder kan læses fra databasen og ind i programmet.
- Fundet ud af at man kan tilgå en liste som et array.

I dag d. 22-02-06

- De resterende kontrol-klasser er lavet → Formue og Pension.
- Kunde kan nu loade Indtaegter (indtil videre Aktieindkomster).
- Vi kan printe 'helt ned' til Aktieindkomster.
- Jes kom 'up-to-date'.
- Skrevet lidt til database omkring fremtidig opgradering af database.
- Skrevet at vi formentlig afgrænser os fra at arbejde med 'sync' af database i denne omgang.
- Databaseklassen er implementeret som en Singleton.
- Foretaget små-rettelser i databasen (OverskudAfSelvstaendigVirksomhed).

I dag d. 23-02-06

- Vi kan nu gemme i DB, gennem save til DB funktion (indtil videre kun på SaveAktier i IndtaegtsKontainer)
- Vi har lavet load funktioner til aktiver og passiver.
- Lavet et udkast til splashscreen til Login skærm.

I dag d. 24-02-06

- Vi har afholdt fredags Briefing (Der blev tændt lys. . . og der blev tændt en flamme)
- Vi har snakket afgrænsninger. Vi fandt ud af at det absolut vigtigste krav til systemet er implementeringen af en 'Print' funktion.
- Der er tilføjet information og overvejelser på baggrund af fredagsbriefingen i et referatdokument.
- Rapport skrivning.
- Database problematik.
- Krav er opdateret i henhold til info fra fredags briefingen.
- Formål er oprettet og skrevet på.
- Researchet info til printning i C# .NET
- Der er besluttet at fremsende en Latex kompileret rapport til vejleder Pia.
- Planlægning af kommende 3 uger er påbegyndt.

I dag d. 27-02-06

- Notesystemet: Afgrænsning / en mulighed for at 'shine' systemet lidt op. Kode:
- Funktionalitet på Kundedata er lavet og virker (fra GUI til DB)
- Slet, Rediger og Tilføj
- Personliste
- Postnr funktion
- Navigeringsknapper
- Sessionsvalg form tilføjet inkl. Kontrol klassen.

I dag d.28-02-06 Kode:

- Påbegyndt oprettelse af ny session (sessionsvalg)
- Designet strukturen af og påbegyndt fremsendingen af sessionsobjektet

Haft besøg af Pia hvor vi har snakket om:

- Estimering og hvordan man kan blive bedre til dette
- Vores fremsendte preview og de kommentarer Pia havde.
- Omfang af studieområde og dokumentation.
- Afgrænsning af vores projekt

I dag d. 01-03-06 Kode:

- Fundet en perfomance-mæssig fejl i postnummer funktionaliteten. Programmet hentede samtlige postnumre i databasen hver gang kundekontainerobjektet blev oprettet, dvs. hver gang en session blev oprettet. Nu er PostNr klassen implementeret som en singleton og kaldet foretages i session; der er nu kun ét databasekald til postnr pr. programstart, resten udføres i programmets datastruktur.
- Opstartsdelen fungerer nu efter hensigten, i denne omgang.
- Sessionen oprettes først ved tryk på 'btnOK' og ikk rdoNySession som før.
- Raadgiver tekststrengen sendes til sessionsobjektet.
- KundeDatadelen fungerer nu efter hensigten, i denne omgang.
- Notefeltet opdateres med noten fra sessionsobjektet ved formload.
- Notefeltet gemmes ved tryk på navigeringsknapperne.
- Der kan skiftes mellem personerne i sessionen.
- By hentes frem til teksfeltet hver gang formen opdateres.
- SQL-kald i Sessionskontainer er opdateret i forhold til Default/Empty

I dag d. 02-03-06 Kode:

- Rettet SaveKunde så den ikke peger på default-session, men på den rette session.
- Lavet polymorfi i Indtaegter, til at gemme indtaegter.
- Identificeret at der er et/flere problemer med load funktionerne (indtaegter)
- Funktionalitet til IndtaegtForhold (Note, navigeringsknapper, personliste)
- Fredagsbriefingsmenu er lavet og udsendt.
- Begyndt på TreeView til at vise indtaegtsforhold.

I dag d. 03-03-06

- Afholdt fredagsbriefing hvor vi informerede om ugens arbejde og næste uges fokus. Derefter så deltagerne, Dion og Johann, en fremvisning af systemet med den funktionalitet det har indtil videre.

  Rapport
- Bearbejdet dele af Pias kommentarer til rapporten (Principper, database, problemformulering, prototyping etc.)
- Skrevet til Database dokumentet omkring
- Skrevet til Design Patterns omkring

  Kode
- Rettet en fejl i gem-algoritmen til Indtaegter
- Rettet småfejl i forbindelse med fremvisningen

I dag d. 06-03-06

- Møde tirsdag omkring kvantificering af krav er booket til kl 14

  Rapport:
- Der er researchet til GUI emnet (lidt gestalt bla). Ud af de 10 'Bud for GUI' er der fundet dem der er relevante.

  Kode:
- Der er arbejdet på printfunktionalitet.
- Crystal Reports har stor funktionalitet, vi skal lige have taget på det.

- Der er blevet arbejdet på treeview.
- Alle slags indtægter kan nu tilføjes.
- Der kan slettes alle slags indtægter.

I dag d. 07-03-06

- Ryddet op i strukturen/placeringen af klasser i Visual; Alle model klasser er samlet i en mappe 'Model' med undermapper til 'Aktiver' etc. Flyttet 'Login' og 'Sessionsvalg' op i 'GUI'.
- Undersøgt teori omkring muligheden for brug af 'abstract factory' designpattern.
- Undersøgt mulighed for brug af Crystal Reports, herunder: Mail korrespondence med Klaus Cohn omkring koden bag en Crystal Report. Korrespondence på eksperten.dk omkring mulighed for at serialize et objekt til XML og dermed lade CR generere ud fra XML dokumentet.
- Afholdt møde omkring kvantificering af krav, med Dion, Kim og Claus P.

I dag d. 08-03-06

- Skrevet noter ind fra mødet med Kim, Dion og Klaus P vedr. Kvantificering af krav.
- De nye fremkomne krav er skrevet ind i kravsdokument i 2.hånds mappen.
- De nye krav er prioriteret.
- Der er udarbejdet et udkast til den rapporten, som systemet skal udprinte.
- Der er arbejdet videre med Crystal Report, kan stadig ikke få data ud af datastrukturen.
- Der er blevet researchet på at eksportere objekter til en .XML fil ved hjælp af serialize funktionen. (Hermed regner vi med at vi kan få skabt kontakt mellem vores Crystal report og dataen i datastrukturen.)
- Der er blevet afsendt en mail til Claus Cohn for at booke et møde med ham for at få løst vores problem med at få kontakt mellem Crystal reporten og vores datastruktur.

- Problemer med treeview. Treeview indeholder knuder af strings. Strings ligner hinanden og man kan risikere at slette det forkerte, da de ikke er sammenlignelige med vores objekter. /*Der kan kun vises tekst, og IKKE tal.*/
- Problem med arv. Ved at sammeligne f.eks indtægt og aktieindtægt, så vil sammenligningen sige de er ens, da aktieindtægt(subklassen) også er en indtægt (superklassen). Dette er et problem.

I dag d. 09-03-06

- Hentet litteratur omkring design af brugergrænseflader og brugervenlighed på Nbs bibliotek.
- Fredags briefing afholdt.
- Opdateret, opstartet og skrevet videre på diverse dokumenter.
- DB normalisering
- Prototyping.
- Keywords sat ind i problemformuleringen
- Design Patterns
- Udvidelses muligheder
- GUI 2. hånds opstartet
- Referat fra sidste Pia vejledermøde gennemgået og rettelser lavet rundt omkring Udkast til printrapport er rettet.
- Der er ryttet lidt, men ikke fyldestgørende op i dokumentstrukturen
- Databasen er opdateret (normalisering, fjernede redundans)

I dag d. 10-03-06

- Hentet litteratur omkring design af brugergrænseflader og brugervenlighed på Nbs bibliotek. Har reserveret en bog mere (online), som skal hentes.
- Lavet er regelset for hvornår elementer på de enkelte skærme skal være synlige og brugbare. Disse regler skal implementeres.
- Udsendt invitation til fredags-briefing i morgen.

- Ændret 'btnIndtaegterOK' til 'btnIndtaegterTilfoej' og fjernet dispose fra denne knap. 'Anuller' er omdøbt til 'Afslut'. Hermed kan man tilføje så mange poster man vil uden at skulle forlade skærmen.
- Lavet funktionalitet til treeview der loader listen og add'er summer på de enktelte indtægts-typer.

I dag d. 13-03-06

- Ændret sessionvalg til at vise sessionens kunders navne samt rådgiver, i stedet for dato og rådgiver. Dette er efter ønske fra Dion.
- Der er skrevet til udvidelsesmuligheder omkring sessionsvalg.
- Der er skrevet kommentarer til koden i kunde, kundekontainer, session, sessionkontainer, dbaccess, kundekontrol, indtaegtkontrol, splashkontrol og sessionsvalgkontrol.
- Der er oprettet et dataset til Crystal rapporten.
- Implementation af GUI regler er påbegyndt

I dag d. 13-03-06

- MVC: Fra Kontrol delen af KundeKontrol.cs er funktionalitet samlet i funktioner og flyttet til View delen Kundedata.cs.
- Der er lavet funktionalitet til rediger/opdater knappen på indtaegter (popup).
- Der er lavet GUI regler for KundeData og Pension.
- Der er skrevet teori omkring Gestalt-lovene i Designafbrugergrænseflade.odt

I dag d. 15-03-06

- Der er arbejdet videre med kommentering af koden.
- Det går bedre med Crystal Report, der mangler dog stadig en del arbejde, specielt med formateringen.
- Vi har nu fået listet alle indtaegtsforholdende med deres summer.
- Ikonerne til Slet, Rediger og Tilføj er færdige og sat på knapperne.
- Der er skrevet lidt til prototyping.

- GUI reglerne er mere eller mindre færdige, der mangler kun Slet og Rediger i Indtaegter der bliver ved med at være synlige (en bug) og tabulator-rækkefølge.
- Der er udsendt invitation til fredagsbriefing i morgen kl 10.

I dag d. 17-03-06

- Afholdt fredagsbriefing.
- Skrevet til prototyping.
- Lagt materiale fra .odt til latex.
- Kigget på tab-rækkefølge, der er dog problemer med niveauerne i dette.
- Kigget på design af skærmbillederne.

I dag d. 20-03-06

- Lang dag på kontoret.
- Vi har samlet alle 2.hånds dokumenter op og smeltet dem sammen i Latex rapport.tex. Inklusiv en del 1.Hånds dokumenter fra 'noter'.
- Alt vi skriver fremover puttes direkte ind i latex.
- Vi er oppe på 54 sider pt. (noget skal skrives sammen, noget skal kortes ned og noget skal uddybes).
- Indledningen af rapporten er tæt ved at være på plads.
- Der er skrevet noget på polymorfi.
- Prototyping er næsten færdig. Skal dog læses igennem igen.
- Der er skrevet til Design Patterns.
- Der er skrevet til Brugergrænseflade.

I dag d. 21-03-06

- Træt dag på kontoret.
- Der er skrevet på Test dokumentet. Vi har stadig ikke planlagt vores session, eller indbudt til den.
- Vi har noget af den forløbige rapport.
- Der er noget fundet noget gengangeri i Brugergrænseflade studieområdet mellem 'Jes_GUI_notes' og design_af_brugergrænseflade. Der skal ryddes op i det.

I dag d. 22-03-06

- Lang dag på kontoret.
- Der er tilføjet et 'eksempel', som skal danne basis som eksempel gennem hele rapporten.
- Der er skrevet på Design Patterns.
- Der er skrevet på Design af Brugergrænseflade.
- Der er skrevet til studieområde.
- Der er skrevet note til Kravslisten omkring krav 41-45.
- Der er tilføjet til Testdokumentet, (testet GemAktiv()) og på test session.
- Der er rettet gennemlæsningsfejl i rapporten.

I dag d. 23-03-06

- Træt dag på kontoret.
- Klaus 29 års fødselsdag!
- Der er skrevet videre på GemIndtægt-eksemplet.
- Der er skrevet lidt på testsessionen
- Vi har booket møde med Kim, Dion er forhindret pga. forvreden ankel.
- Der er skrevet til databasen, (sikkerhed, transaktionsstyring).
- Flere punkter er pillet fra hinanden og sat ind i passende steder i rapporten.

I dag d. 27-03-06

- Træt dag på kontoret.
- Der er skrevet ind på test session.
- Flere andre steder er der skrevet.
- Dette var en flad mandag, og du behøver ikke at . . . træthed!

I dag d. 28-03-06

- Metodologi er skrevet færdig.
- Der er skrevet mere til punkterne i udvidelsesmuligheder, færdig (chr).
- Udprintsfunktionalitet er skrevet, og er færdig som jeg ser det (chr).
- Den røde tråd er i gang.
- Strukturen i rapporten er opdateret og rettet til med kapitler.
- Vi har sendt vores rapport til Pia, hun vil læse den i dag og give kommentarer senere i dag.
- Vi har fået feedback fra Kim og Johann på det de har læst (udgave pr. fredag i sidste uge)
- Der er skrevet til test-session.

10.5 Uge planer

Planen for uge 5 Plan

Mandag:	Dagen bruger vi til at lave lofi. Start med at kigge på regneark og andet materiale som vi har modtaget. Uddelegere opgaven så vi kan lave 3 områder. Målet med denne session er dels at forstå EIKbanks arbejdsgange når det kommer til indtastning/forberedelse af et møde med kunden. Dels at indsamle krav til det nye system.
Tirsdag:	Ser vi gerne Kim til session. Sessionen skulle gerne starte kl. 9:30 og vare 1 time.
Onsdag:	Ser vi gerne Dion til session. Sessionen skulle gerne starte kl. 9:30 og vare 1 time.
Torsdag:	Ingen session. Der skal evalueres på de info vi har fået og der skal forberede til fredagsmødet.
Fredag:	Afholder fredagsmøde, og planlægger næste uges session.

Mål og planer for uge 6 Vi skal have diskuteret vore mappe struktur og placeringen af vores filer. Der er en reel risiko for at miste overbliket over alle vores noter. Den nye prototype skal udvikles. Der skal afholdes en session med Kim, Dion og Claus, hvor vi får fastlagt hvilket felter og informationer der skal være i indtastningsdelen. Vi skal starte på at indsamle informationer om hvordan præsentationsdelen/beregningsdelen udformes. Ide: For at indsamle krav til præsentationsdelen af applikationen kunne der afholdes en brainstorming session / møde med Dion og Kim. Vi skal skære ind til benet og få de vigtige ting på banen. Hvad skal en præsentation for kunden

indeholde. Hvilke grafer og på hvilke tal skal graferne laves. Vi skal kigge nærmere på identificere den hårdeste nød - dette skal vi komme nærmere ind på. Vi skal kigge på et udkast/vores tanker om problemformuleringen. Vi skal undersøge om det er planen at der bliver rene indtastnings opgaver for sekretærerne. Vi skal nærmere et studieområde i forbindelse med rapporten. Vi skal (indtil videre) have fastlagt de principper vi arbejder efter, hvilke ligger grund for vores sytemudviklingsmetode. Kontorstol og telefon efterspørges.

Den hårde nød På dette stadie er det svært at definere et enkelt område som den hårdeste nød, da vi er i en proto-typing fase. I denne fase er det essentielle at komme frem til 'de rigtige' krav, ud fra det input vi får fra virksomheden, hvorfor man kan sige at kravspecifiseringen er den hårde nød.

Den umiddelbare mest risiko-fyldte næste ting vi skal igang med er tilgangen til databasen. Dette er risiko-fyldt i forhold til om det kan komme til at virke på EIKs systemer hos SDC, ligesom at det er risiko-fyldt i forhold til om vi konstruere tilgangen rent teknisk set.

Overordnet for projektet er der andre risiko-områder, som alle er kodetekniske problemstillinger. Nogle af disse er fremtidige og kan opsummeres som:

Udprintningsmodulet Præsentationsdelen, herunder grafisk visualisering i form af grafer etc. Fremskrivningsdelen

Mål og planer for uge 7 Vedtage konventioner for navngivning til brug i kodefasen. Påbegynde design (klassediagram osv.) Forberede test af DB hos Johann. (evt. i form af en mini-applikation)

Forslag til design fremgangsmåde

Identificere objekter ud fra prototypen (hvad består objekterne af, er der del-objekter, etc.?) Klassediagram. Databasediagram. MVC opbygning.

Mål og planer for uge 8 Arbejde videre med design og implementation.

Design Klassediagram og databasediagram Det skal besluttes om vi vil have mere diagrammering

Implementation Der skal udvikles brugergrænseflader (winforms) til de involverede skærmbilleder. Den klasse der skal håndtere tilgangen til databasen skal gøres klar/færdig Funktionaliteten i indtastningsdelen skal påbegyndes. Kontainer klasser skal kodes. MVC arkitekturen skal indkorporeres. Der kunne med fordel laves en test-klasse på dette tidspunkt.

Projektplan for ugerne 10 Dagsorden

Kvantificering af krav med Kim og Dion. Forberede møde. Book et møde til tirsdag. CRUD (create/read/update/delete) funktionalitet. Skrive noget om polymorfi. Research GUI / print. Kode Print funktionalitet. GUI funktionalitet. De 10 gode råd. Treeview. inkl. fejlcheck. Lav en algoritme.

Projektplan for ugerne 9, 10 og 11 Uge 9 Lav research på udprint-funktionalitet Undersøg og påbegynd bearbejdning af studieområde. Fortsæt udviklingen af det nuværende system.

Uge 10

Uge 11

Mål der skal nåes/Ting der skal laves. Sammenkoblingen af view og model. Funktionaliteten til at udskrive noget simpelt. Studieområde med fokus på gui og udskrifter.

10.6 Møder med EIK Bank

Møde med Dion, Kim og Johann, eksempler på Excel ark, d. 19-01-06 Vi gennemgik 2 eksempler på Excel ark, som Kim havde taget med til mødet. I disse eksempler så vi hvilke informationer der indgår og hvilke hovedpunkter vi kan dele en kundes økonomi op i. Herunder er Formue-, indtægts- og pensionsforhold, samt forsikringsdækninger.

Alle variable input (renter, skat etc.) taster rådgiveren/brugeren selv ind.

Applikationen er et status- og simuleringsværktøj, der viser konsekvensen af eventuelle ædringer i en kundes økonomi. Endvidere letter applikationen muligheden for at visualiere dette overfor kunden.

Applikationen repræsenterer et 'her-og-nu' billede af en kundes økonomi. Som det er nu har de simple regneark og mangler overblik over deres materiale.

Systemet skal kunne udvides, også efter at vi er færdige med projektet. Dette er et argument for at lave systemet modulært opbygget og med lav kobling.

Vi fik overordnet information om SDC og hvilket engagement EIK har med dem.

Overordnede spørgsmål til Dion og Johan, d. 23-01-06 Har EIK præferencer i forhold til hvilket sprog applikationen udvikles i? Nej.

Login-funktionialitet? Ja, ikke afdelings-vis, man skal kunne give adgang.

Nuværende arbejdsgang? Dion sender skabelon og materiale. Data ligger i dag på delte netværksdrev, i private kalendersystemer (individuelt) og i de enkelte rådgiveres hukommelse.

Mulighed for central DB? Ja, på nuværende fil-server. DB med stamdata, sessioner etc. Skal EIK indkøbe software i denne forbindelse? (nej)

Hvilke info har EIK private banking afdelingen på kunderne (stamdata etc.)? Dion sender skabelon og materiale.

Skabelon/udspecificering af formue-, indtægts- og pensionsforhold samt forsikringsdækninger ? Dette tager vi ud fra skabelonen som Dion sender og fremtidige interviews.

Note-system i applikationen? Dion foreslår note-historik for hvert punkt (formue, indtægt, etc.) og vi foreslår envidere en generel note-historik for kunden. Sessioner. Der skal være muligheder for at gemme resultatet af en session med en kunde og derefter kunne finde den frem til f.eks. næste møde med kunden. Kontaktforhold fra EIK til os. Vi har sendt en mail rundt til de relevante personer med Klaus's email (klaus_edwin@hotmail.com).

Video med Dion d. 25-01-06 Bearbejdede observationer, Video med Dion 25012006

Efter dette afsnit kommer de oprindelige notater.

Vi har valgt at inddele informationen vi har indsamlet i 5 kategorier efterfuldt af en kronologisk beskrivelse af arbejdsgangen vi så. Sammen med videooptagelserne giver det er godt billede at opstille første LO-FI prototype.

Arbejdsgangens flow: Der er enighed om at flowet i arbejdsgangen forløber i 3 stadier:
1.Oprettelse af kunden.
2.Forberedelse af møde med kunden.
3.Afholdelse af mødet med kunden.
1 og 2 ønskes lejlighedsvis at afholdt samtidigt.

Katagorisering: 1, 2, 3 og 4 Krav og ønsker fra EIKs side.
5 Krav, forståelse, problemer rettet mod vores applikation

Forløbet
1.Fleksibilitet i systemet. Et primært krav/ønske til systemet. Der skal kunne tastes ind og rettes i data, både før og under selve mødesituationen. Systemet skal være et godt værktøj, og assistere konsulenterne.

2.Professionalisme / visualisering i systemet. Endnu et primært krav/ønske til systemet. Det ser ikke professionelt ud når konsulenten sidder med kunderne og roder med et løst Excel-ark. Yderligere skal de økonomiske implikationer og ændringer i disse overskueliggøres overfor kunderne.

3.Kunde relaterede noter. Vigtigheden af et notesystem i applikationen blev tydeliggjort. Underopdeling af noter i kunde og EIK relaterede noter.

4.EIK relaterede noter. Vigtigheden af et notesystem i applikationen blev tydeliggjort. Underopdeling af noter i kunde og EIK relaterede noter.

5.Herunder problemer i og forståelse af nuværende arbejdsgang og krav, problemer og noter rettet mod udviklingen af vores applikation.

1.Fleksibilitet:
Der er forskel på konsulenternes/rådgivernes erfarings-grundlag og begge skal gerne kunne bruge systemet. De erfarne vil gerne have et system der er fleksibelt og som assisterer dem. De uerfarne vil have gavn at et mere strengt og dikterende system Rådgiver er som sådan indifferent overfor hvilket værktøj han bruger, så længe at det er bedre end det han har nu. Det skal være indtastningsvenligt og gøre det hurtigere og nemmere at korrigere i regnskabet. Nogle kunder vil have et mere detaljeret overblik, andre ikke.

2.Professionalisme og Visualisering:
Procentfordelingen af værdien af de eksisterende aktier er vigtig og må gerne kunne udbygges til f.eks. en lagkage-graf. Det primære problem er at de ikke kan vise det værktøj frem de har i dag, da dette ikke ser professionelt ud. (Jeg, Klaus kan se problemer i deres arbejdsgang. Dion mener ikke at arbejdsgangen er noget problem.)

3.Kunde relaterede noter:
Der skal være mulighed for at indtaste og danne sig et overblik over kundens personlige forhold (børn, arvinger, skilsmisse, nuværende arbejde, kroniske sygdomme). Specielt hvis det er forhold der har indflydelse på økonomien. f.eks særeje. I tilfælde af skilsmisse om der skal der betales hustrubidrag el. ej. Børn fra tidl. Ægteskaber. (Udgifter der skal deles eller om det er samlet). Arveret. Hvem arver hvad i tilfælde af død. Oversigt over kundens ønsker i fremtiden. (eks. Ønske om at flytte til udlandet f.eks)

4.EIK relaterede noter:
Noter omkring uklarheder: Er der på baggrund af de økonomiske dokumenter nogen uklarheder og uoverenstemmelser der skal opklares med kunden. Typisk opstået inden egentlig møde med kunden, pga ufuldstændige oplysninger. Det skal være muligt at se hvem der har oprettet en sag, altså hvem der er 'inde i sagen' og normalt har kundekontakten. Hvem der har lavet hvad i systemet.

5.Krav/problemer/forståelse og noter relateret til vores applikation.
Der regnes i hele tusinder. Dette er i henhold til SPT metoden. Som søgekriterier foretrækkes navn, da kunden ikke skal behandles som et tal. Ekstra søgemuligheder ville dog også være fint. Oversigten over pension, formuesammensætning, indtægtsforhold skal kunne deles på alle personer i forholdet(ægteskab, parforhold mv.) Pensioner har et nr. Dette bruges til at kende forskel på flere pensioner i samme pensionsselskab. Giver også mulighed for at søge på dem. Depotindhold. Der skal være en summation så det let at tjekke om alle forhold er indtastet. Der skal være link til f.eks. told og skat. Dette bruges når det er ejendom skal værdisættes. Hvis der er særregler eller udgifter der skal deles, eller om det er samlet. Der skal være mulighed for at taste om der er noget i et parforhold som kan få indflydelse på økonomien. Dette kunne være børn fra tidligere ægteskaber, særeje eller ikke. Det skal være muligt at se hvem der har oprettet en session, altså hvem der er 'inde i sagen', hvem der har kundekontakten. Det er et problem at formler er i et separat ark. Mulighed for at notere præmier på pensioner.

Pension ordninger skal se i en oversigt som er årlig. Nedsparingsoversigt skal være årlig. Opsparings oversigt 25-30år kunne deles i 5 årige perioder. Så man kun ser hver femte år.

Invalide dækning er en oversig HER OG NU. Den skal ikke fremskrives. Det er også et problem at de regneark de sidder med, der kan man komme til at slette forkerte ting.

Selve forløbet: Dion starter med at åbne et eksisterende regneark fra en tidligere session og bruger dette som skabelon til den nye sag/kunde. Dette kræver at han først manuelt sletter de indtastede værdier i det eksisterende ark.

Starter med at undersøge kundens (Henning og Lise) formueforhold. Dion undersøger det udleverede materiale (fra Kims korrespondence med kunden) og kigger på de relevante papirer. (Skatteopgørelser, bank-oversigter, kreditforeningslån, pensionsoversigter). Han taster de relevante total beløb ind for at komme kunne komme frem til kundens formueforhold. Da intet andet var angivet går Dion ud fra at Henning og Lise har fælles økonomi. Indtastningen foregår en enkelt person af gangen. Formuen må i systemet gerne være person-specifik. Dion tjekker at summen på de opgivede tal stemmer overens med del-beløbene og finder ud af at der mangler oplysninger. Formueresultatet tastes ind før og efter afgift (60%, når man får udbetalt sin pension før man går på pension). Links til andre sider bruges til at finde manglende/ekstra info, i dette tilfælde kunne Dion se at kunden bor i en ejerlejlighed, men der er ingen info om denne. Derfor går han på toldskat.dk og ud fra adressen finder lejlighedens værdi, denne regnes med i kundens formue. Indtaster aktiver(Egenkapital, pensioner, fast ejendom, værdipapirer, kontanter og andet) og passiver(det kunden skylder, gæld/lån). Indtaster indtægtsforhold, individuelt. Nu har rådgiveren et overblik over kundens økonomi og gemmer derefter arket på netværksdrevet. Der er intet sikkerhedsproblem i dette, da alle ansatte i realiteten er under samme tro-og-love erklæring. De enkelte pensioner udspecificeres nu i en ny fane i regnearket, hvilket kræver at rådgiveren (igen) først skal slette de allerede indtastede værdier. Herefter skal rådgiveren have fat i et regneark med formler for at kigge på det fremtidige aspekt. Dette ligger under Lotus Notes og er mere eller mindre besværligt at finde frem. Ikke optimalt. Formlerne udfyldes og summerne udregnes. Der indtastes værdier for kapitalforhold (pension) før og efter afgift (40%). Rådgiveren kan nu vurdere fremtidige pensionsforhold ved kundens alder på 65. 'Jo tidligere man går i gang med at betale til en pensions-ordning, jo mindre skal man spare op på grund af rente-effekten'. Pensionsordninger vises generelt på årsbasis. Ved vurderinger omkring 20 år frem vil man normalt kigge på de øknomiske forhold i 5 års intervaller. Rådgiveren kigger nu på livforsikringer/invadilitetsdækning. Disse substitueres af førtidspension fra det offentlige, som modregnes i forhold til den indkomst man har. Dette forhold har ikke ændret sig i meget lang tid og kan derfor betragtes som statisk. Dion har letlæselig bog om problematikken. Vi kunne måske tilføje automatisk udregning af statens udbetaling i tilfælde af invaliditet. 'ville være genialt'.

Noter til Lo-Fi prototypen med Kim, d. 31-01-06 Kunde data
Savnede et tastetur. Men gik hurtigt med på legen. Så at Henning og Lise var

gift. Og at de har børn. Misforstod radioknapperne. Det skulle måske være en checkbox. Med hensyn til barn skal det nok være en checkbox. Brugte han noterner til at fortælle hvem der har sagen. (Klaus tror/mener at kim bruger note feltet ud fra hvad der stod i feltet). Der var tvivl om hvordan Tilføj Person virkede. Fandt hurtig navigationsknapperne og brugte dem flittigt.

Formueforhold Rest skat hedder Latent skat. Latent skat = brutto værdien af depot værdien minus 60På listen over formuer mangler de en total sum og sum pr. person. Han springer mellem de forskellige skærme som han lyster. (Det er hans måde at arbejde på). Kender ikke til købs prisen på aktierne. Indtastede nuværende pris i stedet for. Vi har misforstået kurs. Vi kunne lave kurs pr. aktie, eller en samlet pris, hvilket hedder kursværdi.

Børs Kursen er det kurs som aktierne står til lige nu på Børsen. Kursværdi = antal aktier * kurs pr aktie. Altså hvor mange penge aktierne i det pågældende selskab er værd.

Ønskede et før og nu billede på aktier, så man kan se tab/vind over tid. Kim overser, at der er tilføj knapper både til aktiver og til passiver. (Klaus mener, at dette kunne afhjælpes med groupboxes). På skærmbilledet 'tilføj nyt aktiv' som hører til Formueforhold, der fovirrer knappen pensionordning. Den knap skal ikke være der. Pensionsordning på formueforhold (opdatere) forvirrer. De bør ikke være der. Pensionordninger bør selv opdatere aktiver. Når navigerings knapperene bruges så skal de øvrige sider opdateres. Overskydende skat er overflødig og er blevet fjernet. Liste over aktiver mangler summer. Pensioner kan være splittede, men behøves ikke.

Indtægsforhold Der var tvivl om skærmbilledet. Bruger udelukkelse metoden. Radio knapperne var ikke forstået. Overskydende skat unødvendig. Renteudgifter mangler. Vil gerne taste det hele ind på en gang. Er dette hensigtsmæssigt. Kim leger med selv om at ikke alting virker. :) Kim afprøver opdaterer, men forstår ikke hvad det skal bruges til. Bruger renteindtægter med negativt fortegn for at få sine rente udgifter. Og så på den forskerte person. Renten skal nu ændres (pga forkert person) ved at flytte noget af renten over på Henning. Hvilket ikke var noget problem da de er gift og har fælles økonomi.

Pensionsforhold Kim forstod ikke livsvarigt og/kontra antal år på samme side. Vi har ikke forstået at der er 3 forskellige pensioner: Rate pension (10-25år), engangs og livsvarige. Der skal byttes om på pensionstype og selskab. Der mangler Depotværdi, som af sig selv skal opdatere aktiver Depotværdi er den opsparing der er på pensionen i dag. Liste af pensioner mangler sum om forventet beløb på pensionstidspunktet. Og depot sum, sum ved invalidedækning og sum af dødsdækning. Der mangler en total sum. Ville gerne kunne se begge personers pensioner, men er ikke nødvendig.

Generelt Vi havde forstillelt os at indtastning og rapporting/beregning skulle ses som 2 forskellige ting. Oplysninger om børn er på 'nice to know' basis, men ikke mere. Ikke alle data skal printes. Formueforhold er et over-

bliks billede...

Ekstra noter Vi skal have lavet en gennemgang af de enkelte punkter der skal indtastes. Dette er dog detaljeringsarbejde og vi fokuserer nu på overblikket. STATUSSIDE Problematik. Skal indeholde Aktiver, Passiver og en 'Sum'. (AKT+PAS=SUM) Kim lavede en skabelon over hvad der skal med i en statusside. Man skal kunne se Mand, Kone og Samlet. Der skal være overblik over aktiver, passiver, indkomst og pension årligt. Pensionsoversigt: Hvordan ser det ud når jeg/vui bliver gamle. Her skal vises både før og efter justeringer. Hvordan skal man mest optimalt : Hente eksisterende kunder og tidl. sessioner.

Møde med Klaus Pommern og Johann, Bekymringer omkring S-DC, d. 31-01-06 Vi fandt ud af følgende på mødet. Indtastning / justering skal være mulig hos kunden. Dette pga af at der ofte vil være rettelser og tilføjelser ude hos kunden. Det vil også gavne fleksibiliteten i systemet. Der skal selvfølgelig også være mulighed for at lave justeringer i kundernes økonomi i henhold til deres ønsker. Dette kræver nok at vi laver en lille 'buffer-DB' på applikationssiden, hvor data opbevares indtil der kan synkroniseres med 'moder-databasen'. Vi fik slået fast at det ikke er en mulighed at få SDC til at sætte en server op på dette stadie i projektet. Det er alt alt for dyrt. Aspektet backup skal medtages i overvejelser omkring data opbevaring. Fremtidsmuligheder: Systemet skal naturligvis være modulopbygget således at der eksisterer en mulighed for at have 2vejs kommunikation med SDC database, evt Lotus notes. Vi må i den nuværende situation gå ud fra at vi skal lave en løsning med en Access database på EIK´s fællesdrev.

HUSK: Hold os for øje at vi skal 'crack hardest nut first' = chNf.

En vigtig reflektion på baggrund af vores lille møde: Vi skal passe på med at love for meget. Også selvom vi måske ikke direkte lover noget, men derimod erklærer interesse for aspektet, skal vi passe meget på. Der er meget kort vej fra at erklære interesse til at love. Misforståelsen opstår relativt nemt og det er noget vi skal være obs på. Yderligere skal vi nok lige tage teten op på fredags briefingen og forklare vores tanker omkring dette.

Noter til Lo-Fi session med Dion, Der er også brugt video. D. 01-02-06 OM VORES PROTOTYPING SESSION: Forklare mere om hvad sessionen indebærer. Det skal gå langsomt, Det er ikke ham der er under test. Computeren er dum og reagerer udelukkende på sessionsdeltagerens input. Følge reglerne. Roller. Sessionsdeltageren skal forstå at det kører i et langsomt tempo. Lade forklaringsarket ligger fremme så forståelse for eksempelvis radiobuttons bliver mere tydelig. Skærmbilleder skal gemmes for brugeren. (han må ikke sidde og smugkigge på andre skærmbilleder der ikke bør kunne ses på 'skærmen'. Ikke så meget hjælp til sessionsdeltageren. Sessionsdeltageren blev guidet meget i starten Undervisning om hvordan

computeren bruges.

Generelle noter til/om/fra Dion Dion kom 20 min for sent, så sessionen blev rykket 20 min i den anden ende. Dion var god til at lege med. Dion var alt for hurtig. Dette var et problem for Observator og Computer. Dion sotere papirerne til forskel for Kim. Dvs at Dion indtaster oplysningerne slagvisk. Faneblade til at spinge frem og tilbage er at fortrække. Alle indtastninger skal være Brutto beløb. (Dvs. før skat) Opdatere skal hedde ændre. Mangler noter til pr. aktiv/passiv.

Noter til Kundedata Har ingen problemer med at indtaste person data. Man mangler at kunne indtaste om børn er fællesbørn eller ej. Troede først at en person skulle køres hele vejen i gennem, før man startede på den næste. Det kom Dion hurtigt ud over. Mangler informationer om personen har virksomhed, om han er selvstændig(a/s, aps, el.), eller ej, hvad han arebjder med ol. Dette er for at kunne tale i billeder som kunden kan forholde sig til. Mangler at kunne taster noter til børnene. F.eks. at de havde børneopsparinger, deres navne, cpr ol. Dette kunne gøres i en note til børnene. Børn er ikke en kunde.

Noter til Indtægtsforhold Tilføj systemet er fint forstået. Er der tale om før eller efter arbejdsmarkedsbidrag. Før fortrækkes. Radioknapper er ikke forstået. Overskydende skat skal væk. Mangler Kapital indkomst. Mangler aktieudbytte. Hvor og hvornår de er købt er ret, men ikke nødvendigt. Overser OK og Fortryd til at starte med. Opdatere og tilføj blandes sammen. Kan ikke kende forskel Der mangler ligningsmæssige fradrag. Dion er glad for notesystemet. Opdatere ses som opdatering af skærmen. Derfor skal opdatere hedde ændre. Mangler Pension (sum udbetaling) Der mangler udbetalingsperiode på pensioner. Der mangler en Kapital pension. Kapital pension kan hedde Sum udbetaling. Dion sætter tomme poster ind. Dvs. han siger tilføj → ok uden at taste noget ind. Bruger generelt siden om Skatte beringning/regnskab.

Noter til Formueforhold Mangler link til www.skat.dk så Dion kan tjekke huspriser. Dette er en afgrænsning af kravenede set fra vores side. Til Aktiverne mangler der investeringsbeviser. Dion bruger ikke alle felter når han indtaster aktiver, bank indestående og fast ejendom. Under passiver blev Navn. Misforstået. Skal hedde långiver i sted for. Forældre lån skal hedde Famile lån. Kassekredit skal have Max overtræk, Nuværende overtræk, rente sats og rest løbetid. Kredit lån skal have Rest løbetid, Type af lån (rentetilpasningslån, lo.) og rente sats. Sætter igen tomme poster ind. Ser ikke skærmen som status/overbliksbillede som Kim gjorde det.

Noter til Pensionforhold Der mangler under tilføj Pension 'Bank Ordning' og den skal have Årlig indbetaling, Kvartårlige indbetaling og månedlige indbetalinger, samt hvornår disse indbtalinger startede Undlander at taste alle informationer om en pension. Indsætter tomme poster. Dion skelner mellem bank(prognose afkast) og forsikringspension (garanteret afkast). Vores tilføj passer fint til forsikrings pension. Bank forsikring skal have

felterne. Værdi i dag/reserve, årlig, halvårlig, kvartal, mdr. indbetaling, stigning fra en eller anden dato til så og så meget og/eller en procentsats. Til bankordning (Pension) det ville være ret at kunne se fordelingen af Aktier, obligationer o.l. Både med beløb og forrentning.

Aktie 50.000	forrentning 7%
Obligation 20.000	forrentning 5%
Kontant 100.000	forrentning 2%

Nedkogt materiale om Lo-Fi prototype session 1 d. 01-02-06 Hvad er gået godt?

- Vi skal sørge for at applikationen bliver fleksibel at arbejde med. Altså at den ikke er arbejdsgangdikterende, men derimod arbejdsgangassisterende. Dette er så vidt muligt overholdt:
- Man kan springe frem og tilbage mellem skærmbillederne, samtidig med at programmet gemmer den indtastede information.
- Opdelingen af arbejdsgangen i en indtastnings- og en talbearbejdningsdel lader til at virke fint.

  Hvad skal rettes/ hvilke problemer skal til revision?

- Radioknapper er svært forståelige.
- Både ved personer(hvor man vælger den person, som man taget udgangspunkt i i øjeblikket. Her skal børn kunne tilføjes med en checkboks i stedet.
- Radioknapper ved Pop-Ups er problemet at al information forsøges indtastet på en gang.
- 'Opdater/slet/tilføj' går de ofte fejl af, det er ikke nemt nok at forstå, ikke intuitvt at forstå. Dette kunne måske rettes ved at præcisere hvad knapperne gør (hvad er det der i denne situation bliver opdateret, slettet og tilføjet). Opdatere kunne hedde ændre måske.
- For at opnå endnu mere fleksibilitet skal vi have lavet en sideoversigt nederst på skærmbillederne, som gør det muligt at springe mellem siderne en slags 'faneblads' løsning i stedet for bare 'en side frem/tilbage løsningen'. Når disse benyttes skal de øvrige sider opdateres.
- Notesystemet skal gennemgåes og revideres. Med henblik på hvad det skal bruges til, hvor det skal være muligt at tage noter til. Hvad med historikken: Skal man kunne se tidligere noter.

- Hvem har tilføjet noten med tidsstempel.
- Mulighed for at rette i tidligere noter.
- Overskydende skat skal væk under indtægtsforhold.
- Formueforhold: Under passiver blev Navn misforstået. Det skal hedde långiver i sted for.
- Formueforhold: Forældre lån skal hedde Famile lån.
- Pensions ordninger under Formueforhold -¿ tilføj skal væk.
- Formueforhold → tilføj: Rest skat skal hedde latent skat. Latent skat er brutto værdien af depotværdien minus 60%.
- Formueforhold → tilføj: Kurs er prisen på en aktie. Kursværdi er antal * kurs. Så er der børs kurs er den aktuelle kurs på børsen.
- Pensionsforhold: Der skal byttes om du pensionstype og selvskab.

  Skal fjernes:

- Overskydende skat fra indtægtsforhold.

  Hvad skal tilføjes?

- Alle indtastninger skal være brutto (altså før skat)
- Under indtægtsforhold: Vi bør benytte FØR arbejdsmarkedsbidrag og ikke EFTER arbejdsmarkedsbidrag.
- Det skal være muligt at tilføje noter til Aktiver og Passiver under Formueforhold.
- Det skal være muligt at få et overblik over kunden og deres personlige forhold.
- Har de børn, hvis ja hvor mange og hvad er deres navne og CPR numre. Er børnene fællesbørn eller ej? Er der tilknyttet nogen børneopsparing? Børn skal ikke opfattes som en kunde.
- Hvilken virksomhed arbejder de i, er de lønmodtagere/selvstændige, hvis selvstændige er det Aps. eller er det A/S. Dette er for at kunne tale i billeder som kunden kan forholde sig til.
- Det skal måske være mere synligt hvem der er kontaktperson for kunden.

- Indtægtsforhold: Mangler Kapital indkomst.
- Indtægtsforhold: Aktier skal upspecificeres med kurs de er købt til, antallet og dagens kursværdi på Børsen. Måske info om hvor og hvornår de er købt, men absolut ikke essentielt.
- Indtægtsforhold: Der mangler ligningsmæssige fradrag.
- Indtægtsforhold: Mangler Pension (sum udbetaling) Der mangler udbetalingsperiode på pensioner. Der mangler en Kapital pension. Kapital pension kan hedde Sum udbetaling. Indtægtsforhold: Der mangler Rente udgifter.
- Formueforhold: Mangler link til www.skat.dk så Dion kan tjekke huspriser. Dette er en afgrænsning af kravenede set fra vores side.
- Formueforhold: Til Aktiverne mangler der investeringsbeviser.
- Formueforhold: Kassekredit skal have:
- Max overtræk.
- Nuværende overtræk
- Rente sats
- Rest løbetid.
- Formueforhold: Kredit lån skal have Rest løbetid, Type af lån (rentetilpasningslån, lo.) og Rente sats.
- Pensionsforhold: Der er 3 typer af pensioner. Rate pension (10-25 år), Engang udbetaling og livsvarigt udbetaling.
- Pensionsforhold: Der mangler under tilføj Pension 'Bank Ordning'. Skal have 1/1, $\frac{1}{2}$, $\frac{1}{4}$ årlige indbetalinger, samt månedlige indbetalinger, samt hvornår disse indbetalingerne startede.
- Pensionsforhold: Dion skelner mellem bank(prognose afkast) og forsikrings pension (garanteret afkast). Vores tilføj passer fint til forsikrings pension.
- Pensionsforhold: Bank forsikring skal have felterne. Værdi i dag/reserve, årlig, halvårlig, kvartal, mdr. indbetaling, stigning fra en eller anden dato til så og så meget og/eller en procentsats.
- Pensionsforhold:

- Til bankordning (Pension) det ville være ret at kunne se fordelingen af Aktier, obligationer o.l. Både med beløb og forrentning.

Aktie 50.000	forrentning 7%
Obligation 20.000	forrentning 5%
Kontant 100.000	forrentning 2%

- De forskellige lister af Aktiver, Passiver og Pensioner skal haver koloner med summer. En total og en for hver part i familien.

- Pension skal have beløb på pensionstidspunket, depotværdi (depotværdi = den værdi, invalidedækning og dødsdækning.

- Formueforhold → tilføj: Ville gerne kunne se et før og nu kurs på aktier så man kan se tab/vind over tid.

  Vores indtryk og erfaringer med sessionerne:

- Forklare mere om hvad sessionen indebærer. Det skal gå langsomt, Det er ikke ham der er under test. Computeren er dum og reagerer udelukkende på sessionsdeltagerens input.

- Følge reglerne. Roller. Sessionsdeltageren skal forstå at det kører i et langsomt tempo.

- Lade forklaringsarket ligger fremme så forståelse for eksempelvis radiobuttons bliver mere tydelig.

- Skærmbilleder skal gemmes for brugeren. (han må ikke sidde og smugkigge på andre skærmbilleder der ikke bør kunne ses på 'skærmen'.

- Ikke så meget hjælp til sessionsdeltageren. Sessionsdeltageren blev guidet meget i starten

- Undervisning om hvordan computeren bruges.

Noter fra møde med Kim, 02-02-06

- 'Det system vi udvikler skal være et værktøj der hjælper kunden med at få et bedre overblik over dennes økonomi og relevante fremskrivninger på en variabel måde' - resten er 'nice to have'

- 'Systemet vi udvikler skal bruges til at forstå alle kundens bankmæssige forhold og vel at mærke få kunden til at forstå de økonomiske aspekter i deres nuværende situation'

- At beregne skatter, det ville være fint hvis systemet automatisk tager skat fra beløbene når de skal behandles, eksempelvis et skema der er variabelt efter renteindtægter og antal år efter behov.

- Det blev i øvrigt pointeret igen at vi endelig skal 'forstyrre' hvis der er information vi har brug for.

Alle beløb indtastes som brutto, dvs. før skat (man taler ikke om skat før til sidst).	must have
Formlerne (de 4) skal som sådan ikke med i systemet	nice to have
Det er fint at opremse børne-forhold, men kun for at nævne familie- og økonomiske forhold (børneopsparing, generationer etc.) Børn noteres generelt som en note.	nice to have, Hele familen opfattes som en kunde
Hvis kunden har virksomhed indgår virksomhedens værdi under aktiver (værdi=egenkapital (+goodwill))	must have
Links til anden info betyder ikke meget	nice to have
Systemet skal ikke nødvendigvis identificere rådgiveren over for kunden	nice to have
Identificering af rådgiver internt i EIK er ubetydelig	nice to have
Opsummering af aktiver og passiver	must have
Notesystemet skal bare være simpelt	could have
Udbygget notesystem er	nice to have

- Hver kunde har én rådgiver
- PB rapport laves typisk en gang hvert år.
- Der er en kun én overordnet ansvarlig rådgiver pr. kunde
- Gamle rapporter uvæsentlige
- Der gemmes kun én version af rapporten, da man ellers
- risikerer at forveksle gamle og nye versioner.
- PB rapporten gælder ikke som en 'aftale'

Simulering af fremskrivninger er	must have
Login ja hvis smart for os, ellers	nice to have
Central DB er vigtig i forbindelse med fremtidige muligheder (i fremtiden kan man lave statistik på kundemassen etc.)	could have
Udprint-funktion er	must have

- Status-sidens indhold ændrer sig ikke.
- De dele der kan ændres i er Aktier, Obligation og Pension.

- Forbrugsmuligheder def.: 'Hvornår er dine 10 millioner spist op, hvad kan du tillade dig'
- En note i forbindelse med DB snakken: Skulle det ske at en rådgiver kunne finde på at skifte til en konkurrerende bank, ville han kun kunne tage en kopi af sin kundebase med sig hvis systemet arbejder med en central db. Hvis hver rådgiver derimod kun har deres kundebase liggende lokalt ville han kunne tage hele basen med uden at virksomheden ville kunne få fat på den. Dette er i EIKs interesse da de gerne vil holde på kunderne, med andre ord ligger der et CRM aspekt her.

Session 3 09.02.06

Prototype 2, færdig udgave af indtastningsdelen

NØGLE: Præcisere hvilke felter der skal være i indtastningsdelen i systemet.

Christians noter

Kundedata

Samboende/samlever: Forskel i forhold til pension, men i praksis ingen forskel. Derfor fjernes denne person-status. Information om selvstændig virksomhed m.m. kan fint noteres/lægges i notesystemet. 'Ændr person' omdøbes til 'ændr oplysning' 'Fraskildt' omdøbes til 'Enlig' Information omkring børn kan fint noteres/lægges i notesytemet, fyldestgørende nok. Et felt til 'udlands-dansker' er ikke nødvendigt, denne info lægges i notesystemet. Der behøver ikke være plads til mere end 2 personer i person-listen.

Indtægtsforhold

Hvilke information i listboxen: 'Hans, hendes og en total' Altså en kolonne (lodret) til person 1, en kolonne til person 2 og en kollonne hvor der summeres. I rækkerne (vandret) listes indtægterne. Kim kunne godt ønske sig 6 kolonner i alt, hvor fremskrivninger inddrages. Der bliver dog enighed om at holde sig til 'indtastnings-delen' i denne omgang. Både Kim og Dion er med på workflowet/arbejdsgangen.

Indtægtsforhold, popup

Fradrag fjernes da dette ikke er en skatte-beregning/selvangivelse. Kapital pension og Pensioner fjernes. 'Vi tænker i skat, men skriver ikke noget om det her' så altid: før skat! Posternes grupperes efter type De 4 første grupperes efter typen 'løn' → A-Indkomst (kildeskat på alle 4) AktieUdbytte omdøbes til AktieIndkomst (kursgevinst og udbytte er skattemæssigt det samme) Renteindkomst kommer med 'Skat' tilføjes, denne regner rådgiveren selv ud og taster ind. Der skelnes ikke mellem forskellige typer af pensioner 'Pensioner'

1.Løn indkomst Firma Beløb

2.Offentlig ydelse Ydelse (Der er kun ét beløb, der findes kun førtids- og folkepension og man kun få enten den ene eller den anden samtidigt)

3.Bestyrelses honorar Firma Beløb

4.Pensioner Selskab Beløb

5.Overskud selvstændig virksomhed Selskab Beløb

6.Rente indkomst Beløb (evt. flere indtastningsfelter så der kan regnes sammen, dog prioriteres et enkelt beløb højest for at holde det simpelt)

7.Aktie indkomst Udbytte (når virksomheden værdi-sætter sine aktier) Kursgevinst/tab (når man sælger sin aktie)

8.Rente udgift Prioritetsgæld Bank/anden gæld

9.Skat 2 former: Skat af løn og kapital indkomst (beskattes ens) Skat af aktieindkomst (beskattes lavere)

Generelt: Indtægtsforhold omfatter også udgifter (renteudgifter og skat)

Formueforhold

Fælles for listboxes tilhørende aktiver og passiver: 3 kolonner (Person 1, person2 og total)

Aktiver, popup

Egenkapital er ikke et aktiv, denne fjernes. 'Investeringsbevis' er det samme som 'aktie'. 'Anparter' erstattes med 'selvstændig virksomhed'. (anpart beskattes som en selvst. Virk.) Poster: Fast ejendom Selvstændig virksomhed Aktie Obligation Bank indestående Andre akriver

1.Fast ejendom Bolig Fritids bolig Ervervs ejendom Udlejnings ejendom

2.Selvstændig virksomhed Selskab Personlig KS (Komadit selskab) Her skriver man beløbet ud for den type virksomhed det drejer sig om.

3.Pensioner Der tastes ikke noget ind her, dette hentes fra 'Pensions forhold'

4.Aktie Opdeleles som: Frie midler Pensionsmidler

Her ønsker Kim en lidt anderledes opstilling, da han gerne vil kunne taste alle aktier ind i samme vindue. Dermed kommer der kolonner med Selskab, Kurs, Kursværdig og Antal, samt en markering af hvilken type det drejer sig om (frie midler vs. pension.) Aktierne kan dermed tastes ind nedaf. De frie midler inkluderes i aktiv-massen, dette gør pensionsmidlerne ikke.

5.Andre aktiver Kunst/Antikviteter

6.Bank indestående Beløb

Passiver, popup

Kreditlån(?) fjernes. (Realkredit og prioritetslån er det samme) Andre passiver tilføjes 6 punkter Prioritetsgæld Familielån Kassekredit Bank gæld Andre passiver Latent skat

1.Prioritetsgæld Långiver Type Rentesats Rest Løbetid Beløb

I rente tastes sats og ikke beløb, altså årlig procent sats. Type (Kontantlån, obligationslån og flex lån)

2.Familielån Magen til 1.

3.Kassekredit Magen til 1 og 2. (Max træk etc. er ligegyldigt. Hvis kun 25.00 træk ud af 100.000 grænse, posted 75.000 som kontanter under aktiver)

4.Bank gæld Magen til 1, 2 og 3.

5.Andre passiver Andre former for lån (pantebrevs-gæld, gældsbrev(ikke familie-lån) m.m.) Magen til 1, 2, 3 og 4.

6.Latent skat Opdeles i 4 typer: Pensioner Ejendomme Virksomhed Aktie Pensionsforhold

Informationer i listbox: Der skelnes mellem dækninger: Reelle pensioner Dækning uarbejdsløs Dækning dødsfald. Type Tidsbegrænset

og andre: Livrente Engangsbeløb

Pensionsforhold, popup

Kapital Rate Livrente Forsikring

Indtastningsmulighedern er de samme for alle 4, der hvor de er forskellige undlader rådgiver at taste noget ind (prognoseværdi, beløb årligt).

'Start dato' omdøbes til 'Start Udbetaling'. 'Antal år' omdøbes til 'Udløbsdato'. 'Beløb' omdøbes til Prognose værdi'

Jes Noter

Kundedata:

Information om firmaforhold smides i notefeltet. Der skal tilføjes information om landeforhold. (i tilfældet det er en udlandsdansker) Knappen 'ændre person' bør hedde 'ændre oplysninger'

Indtægtsforhold:

Det handler her kun om indtægt og renteudgifter. I oversigtsfeltet, blev der foreslået at der kunne være 6 kolonner istedet for 3. Dette er dog mere i henhold til fremskrivningdelen af systemet, og er derfor ikke så relevant i denne sammenhæng. Alle fradrag fjernes fra indtægtsforhold. Det er vigtigt at fokusere på at alle poster er før skat. Grunden til dette er at man kan rykke rundt på forskellige parametre og poster inden skatten beregnes. De 4 første poster betegnes A-INDKOMST (lønindkomst, bestyrelseshonorar, offentlige ydelser, pension) De 2 næste poster betegnes som KAPITAL-INDKOMST (overskud af selvs. Virksomhed, renteindkomst) Aktieindkomst (gælder både salg af aktier og den årlige udbetalte præmie). Offentlige ydelser dækker over (folkepension og førtidspension). Dette skal ikek specificeres ud i systemet. Renteindtægt (her kan der være flere → det skal være muligt at lægge flere beløb sammen. En anden løsning er at brugeren selv indtaster summen af disse flere beløb) Renteudgift. Består af: prioritetsgæld (bolig gæld, som har første prioritet) bank/anden gæld (forbrugsgæld) Skat. Består af: Skat (kapitalindkomst og lønindkomst har samme skattesats) Aktieskat (heraf betales mindre skat end ved posten ovenover) 'Skat' posten beregnes ud fra information fra Told&Skats hjemmeside.

Formueforhold.

AKTIVER: Alle poster er grupperet efter likviditet. Dvs hvor hurtigt det kan omsættes til kontanter. Aktier (indeholder både Aps og A/S) Selvstændig virksomhed dækker over f.eks. Anpart i en vindmølle, osv. Der skal her være plads til flere indtastninger. 'Pension'. Her hentes data fra indtastningen i Pensionsforhold. 'Aktier': Indtastningen forgår i følgende skema på ÉN gang.

SELSKAB ANTAL KURS KURSVÆRDI PENSION / FRIE MIDLER

Pension/frie midler er et enten eller felt. Hvis det er en 'pensionsaktie' skal den være en del af den samlede 'pension' post. Formålet her er kun at få et overblik over fordelingen af aktieposter under pensionsforhold. Det er kun 'frie midler' aktier, altså de aktier der er markeret som 'frie midler', der skal indgå under posten Formueforhold → aktiver → 'aktier'. PASSIVER: Kassekredit: Værdien for den maksimale af banken tilladte overtræk (eks. 100.000) kan godt tastes ind som fuld værdi under kassekredit, men så skal forskellen mellem det nuværende reelle overtræk (eks. 25.000) og det maksimalt tilladte overtræk 100000-25000=75000 overføres til aktiver under kontanter.

Pensionsforhold: Der findes 3 forskellige slags dækning: 1.Pension 2.Sygdom 3.Død Der findes 4 forskellige typer pensioner: 1.rate 2.Livsvarig 3.Livsrenter 4.Kapital (engangs) Vores indtastningskema skal ligne det som Kim tidligere har udleveret. Prognoseværdi feltet angiver værdien af pensionen den dag du bliver pensioneret. Altså inklusiv forrentning og fremtidige årlige indbetalinger.

Session 3, 09.02.06 Prototype 2, færdig udgave af indtastningsdelen

NØGLE: Præcisere hvilke felter der skal være i indtastningsdelen i systemet.

Christians noter

Kundedata Samboende/samlever: Forskel i forhold til pension, men i praksis ingen forskel. Derfor fjernes 'samboer' status. Information om selvstændig virksomhed og firmaforhold m.m. kan fint noteres/lægges i notesystemet. 'Ændre person' omdøbes til 'ændre oplysning' 'Fraskildt' omdøbes til 'Enlig' Information omkring børn kan fint noteres/lægges i notesytemet, fyldestgørende nok. Et felt til 'udlands-dansker' er ikke nødvendigt, denne info lægges i notesystemet. Der behøver ikke være plads til mere end 2 personer i person-listen.

Indtægtsforhold Indtægtsforhold omfatter kun indtægt og udgifter (renteudgifter og skat). 'Vi tænker i skat, men skriver ikke noget om det her' så altid: før skat! Det er vigtigt at fokusere på at alle poster er før skat. Grunden til dette er at man kan rykke rundt på forskellige parametre og poster inden skatten beregnes.

Hvilke information i listboxen: 'Hans, hendes og en total' Altså en kolonne (lodret) til person 1, en til person 2 og en hvor der summeres. I rækkerne (vandret) listes indtægterne. Kim kunne godt ønske sig 6 kolonner i alt, hvor fremskrivninger inddrages. Der bliver dog enighed om at holde sig til 'indtastnings-delen' i denne omgang. Opfattelse: Både Kim og Dion er med på workflowet/arbejdsgangen.

Indtægtsforhold, popup Alle fradrag fjernes fra indtægtsforhold, da dette ikke er en skatte-beregning/selvangivelse. Posternes grupperes efter type De 4 første poster betegnes A-INDKOMST = kildeskat(lønindkomst,

bestyrelseshonorar, offentlige ydelser, pension) De 2 næste poster betegnes som KAPITAL-INDKOMST (overskud af selvs. Virksomhed, renteindkomst) AktieUdbytte omdøbes til AktieIndkomst (kursgevinst og udbytte er skattemæssigt det samme) Renteindkomst kommer med 'Skat' tilføjes, denne regner rådgiveren selv ud og taster ind. Der skelnes ikke mellem forskellige typer af pensioner under 'Pensioner'

1.Løn indkomst Firma Beløb

2.Offentlig ydelse Ydelse (Offentlige ydelser dækker over folkepension og førtidspension. Dette skal ikke specificeres ud i systemet.)

3.Bestyrelses honorar Firma Beløb

4.Pensioner Selskab Beløb

5.Overskud selvstændig virksomhed Selskab Beløb

6.Rente indkomst Beløb (evt. flere indtastningsfelter så der kan regnes sammen, dog prioriteres et enkelt beløb højest for at holde det simpelt)

7.Aktie indkomst Udbytte (den årlige udbetalte præmie) Kursgevinst/tab (når man sælger sin aktie)

8.Rente udgift Prioritetsgæld (bolig gæld, som har første prioritet) Bank/anden gæld (forbrugsgæld)

9.Skat 2 former: Skat af løn og kapital indkomst (beskattes ens) Skat af aktieindkomst (beskattes lavere) 'Skat' posten beregnes ud fra information fra Told&Skats hjemmeside.

Formueforhold Fælles for listboxes tilhørende aktiver og passiver: 3 kolonner (Person 1, person2 og total)

Aktiver, popup Alle poster er grupperet efter likviditet. Dvs hvor hurtigt det kan omsættes til kontanter. Egenkapital er ikke et aktiv, denne fjernes. 'Investeringsbevis' fjernes da det er det samme som 'aktie'. 'Anparter' erstattes med 'selvstændig virksomhed'. (anpart beskattes som en selvst. Virk.) Poster: Fast ejendom Selvstændig virksomhed Aktie Obligation Bank indestående Andre aktiver

1.Fast ejendom Bolig Fritids bolig Ervervs ejendom Udlejnings ejendom

2.Selvstændig virksomhed Selskab Personlig KS (Komadit selskab) Her skriver man beløbet ud for den type virksomhed det drejer sig om.

3.Pensioner Der tastes ikke noget ind her, dette hentes fra 'Pensions forhold'

4.Aktie Aktier (indeholder både Aps og A/S) Opdeleles som: Frie midler Pensionsmidler

Her ønsker Kim en lidt anderledes opstilling, da han gerne vil kunne taste alle aktier ind i samme vindue. Dermed kommer der kolonner med Selskab, Kurs, Kursværdig og Antal, samt en markering af hvilken type det drejer sig om (frie midler vs. pension.) Indtastningen af Aktier foregår en af gangen nedefter og på én gang. SELSKAB ANTAL KURS KURSVÆRDI PENSION / FRIE MIDLER De frie midler inkluderes i aktiv-massen (altså

medregnes i den værdi der vises på oversigten på Formueforhold), dette gør pensionsmidlerne ikke.

5.Andre aktiver Kunst/Antikviteter

6.Bank indestående Beløb

Passiver, popup Kreditlån(?) fjernes. (Realkredit og prioritetslån er det samme) 'Andre passiver' tilføjes 6 punkter Prioritetsgæld Familielån Kassekredit Bank gæld Andre passiver Latent skat

1.Prioritetsgæld Långiver Type Rentesats Rest Løbetid Beløb

I rente tastes sats og ikke beløb, altså årlig procent sats. Type (Kontantlån, obligationslån og flex lån)

2.Familielån Magen til 1.

3.Kassekredit Magen til 1 og 2. (Max træk etc. er ligegyldigt. Hvis kun 25.00 træk ud af 100.000 grænse, posted 75.000 som kontanter under aktiver)

4.Bank gæld Magen til 1, 2 og 3.

5.Andre passiver Andre former for lån (pantebrevs-gæld, gældsbrev(ikke familie-lån) m.m.) Magen til 1, 2, 3 og 4.

6.Latent skat Opdeles i 4 typer: Pensioner Ejendomme Virksomhed Aktie

Pensionsforhold Informationer i listbox: Type Tidsbegrænset

Der findes 3 forskellige slags dækning: 1.Pension 2.Sygdom 3.Død Der findes 4 forskellige typer pensioner: 1.Rate pension 2.Livsrenter (Livsvarig) 3.Kapital (engangs) 4.Forsikring Der er her lidt tvivl under noteindtastningen hvorvidt 'livrenter' og 'forsikring' er det samme?

Vores indtastningskema skal ligne det som Kim tidligere har udleveret. 'Beløb' omdøbes til Prognose værdi' Prognoseværdi feltet angiver værdien af pensionen den dag du bliver pensioneret. Altså inklusiv forrentning og fremtidige årlige indbetalinger.

Pensionsforhold, popup Indtastningsmulighedern er de samme for alle 4, der hvor de er forskellige undlader rådgiver at taste noget ind (prognoseværdi, beløb årligt). 'Start dato' omdøbes til 'Start Udbetaling'. 'Antal år' omdøbes til 'Udløbsdato'.

Spørgsmål til møde med EIKBank 07-03-2006

- Hvad er brugervenlighed?
- Fejl-40?
- Styre input i felter? (kun tal i cpr)
- Hvad er nemt er lære?
- Hvor lang tid må det tage for en ny person?
- Hvor lang tid må det tage en erfaren person?

- Hvor mange gange må man prøve/forsøge at bruge systemet før man klare tidskravet?
- Hvad er et professionel look?
- Hvad er et nemt/overskueligt overblik?
- Kunne I komme med et forslag på papir til hvordan data skal stilles op?
- Hvad med layout?
- Skal der printes på brevpapir?
- Hvad er udvidelsesvenligt?
- Med hvad? Nye felter, funktioner(arbejd ude/nye database), typer (pensioner/forsikringer/indkomster oa.)?

Litteratur

[1] D. Avison and G. Fitzgerald. *Information systems Development*. McGraw-Hill, 3 edition, 2003. ISBN 0-07-709626-6. 528 pp.

[2] S. Bennet, S. McRobb, and R. Farmer. *Object-Oriented Systemts Analysis And Design Using UML*. McGraw-Hill, 2 edition, 2002. ISBN 0-07-709864-1. 315 pp.

[3] S. Burbech. Applications programmering in smalltalk-80. pages 1–10, 1992. Kilden er valgt fordi at den giver en god beskrivelse af hvad MVC er og kan bruges, samt rolle fordeling mellem klasser. Burbech er Ph.D og specialist i Objekt Teknologi. Burbech har skrevet Peer to Peer Webservices conference i 2001 (isbn 0-596-00110-X). Smalltalk er et programmering sprog fra slut 1970'erne.

[4] L. Cortes. Designing a graphical user interface. pages 1–10, Maj 1997. MCToday er en site der beskæftiger sig med tekniske og computer relaterede emner med relevans for ansatte med relation til sundshedssektoren. Formålet er at give information, som er brugbar på alle niveauer af computererfaring og for alle typer af akademisk og klinisk praksis. Siten rummer en blanding af originalartikler og referencer til webressourcer. Der findes mange gode sites med information relevant for ansatte med relationer til sundshedsektoren. MCToday forsøger at samle links til lærerige artikler på disse sites. Original artikler skal forsøge at koncentere sig om at dække områder, der ikke er gennemgået i detaljer på internettet endnu. Der opfordres til at bidrage til siten, hvis man har relevant erfaren og ekspertise. De publiserede artikler aflønnes ikke.

[5] A. Dix, janet Finlay, G. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice Hall, 2 edition, 1997. ISBN 0-13-239864-8. 416–418 pp.

[6] N. Panchal. Implementing mvc design pattern in .net. pages 1–4, februar 2003. Kilden er valgt fordi at den beskæftiger sig med MCV i det programmeringsprog vi koder i. Artiklen beskæftiger sig med fordele/ulemper, Principper og eksempler omkring MVC. Panchal har skrevet andre artikler om design pattern på www.VBdotNetHaeven.com Han

arbejdede i 2003 som konsulent i New York med mere end 5 år erfaring i software udvikling. Kilden henviser til : C# Design Pattern By Cooper (Book) Her vi ikke kigget på. http://www.zdnet.com.au/ (Online Resource) Ubrugeligt. http://www.dmbcllc.com (Online Resource) ubrugeligt. http://www.indiawebdevelopers.com (Online Resource) ubrugeligt. http://st-www.cs.uiuc.edu (Online Resource) Den har vi brugt.

[7] C. Torrans and N. Dabbagh. Gestalt and instructional design. pages 1–9, marts 1999.

[8] Unknown. Model-view-controller (microsoft patterns). pages 1–6. Kilden er valgt fordi at den beskæftiger sig med MCV. Kilden heviser til : Model-View-Controller began as a framework developed by Trygve Reenskaug for the Smalltalk platform in the late 1970s [Fowler03]. The version you have just read references the following works: [Burbeck92] Burbeck, Steve. "Application Programming in Smalltalk-80: How to use Model-View-Controller (MVC)."University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive. Available at: http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html. Den har vi brugt. [Fowler03] Fowler, Martin. Patterns of Enterprise Application Architecture. Addison-Wesley, 2003. Den har vi ikke brugt. [Gamma95] Gamma, Helm, Johnson, and Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995. Den har vi brugt.

Figurer

Tabeller