2010

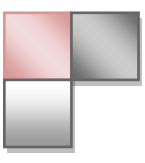
Kursusadministrations-system til BEC

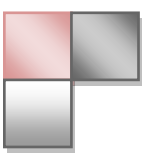
Rapport

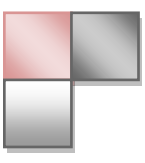
Stine Dahl	_____
Marianne Vesterdal	_____
Martin Reimer	_____
Daniel Falkner	_____

Gruppe 1
28. maj 2010

Indholdsfortegnelse







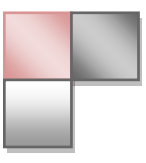
Forord

af Stine

Denne rapport er et 2. Semester tværfagligt eksamensprojekt i fagene software construction og software design. Rapporten er udarbejdet som et fiktivt systemudviklingsprojekt, hvor vi skal lave et kursusadministrationssystem for Bankernes EDB Central (Herefter BEC). Vi har ingen direkte dialog med virksomheden så al "info" er enten rekvireret fra deres meget udmærkede hjemmeside eller vi har brugt vores fri fantasi.

I dette forløb vil vi benytte os af de værktøjer og teknikker som vi har lært de sidste to semestre. I software design er det unified process (Herefter UP) hvor vi læner os meget op ad bogen "Applying UML and Patterns" skrevet af Craig Larman. Derudover diverse kopier udleveret af Lars Christian Kofod.

I software construction består selve koden af JAVA og koden er skrevet i programmet Netbeans.



Problemformulering

af Gruppen

Hvordan får vi udarbejdet en informativ rapport der underbygger konstruktionen af et funktionsdygtigt kursusadministrationssystem til BEC?

Ovenstående formulering giver anledning til følgende punkter:

- Hvilke værktøjer vil vi anvende i processen?
- Hvordan optimerer vi arbejdsformen som gruppe?
- Hvordan sikrer vi at alle gruppens medlemmer får en chance for kompetenceudvikling?

Metoder

af Daniel

Projektetableringen skal give os et grundlag for projektet og hvordan vi organiseres som gruppe. Derefter vil vi lave en forretningsanalyse af BEC som en virksomhed. Forretningsanalysen skal give os et indblik i virksomhedens styrker og svagheder, samt mulighed for fremtidig udvikling.

Vi vil arbejde med Unified Process' faser, Inception og Elaboration, delt op i iterationer og udvikle applikationen og rapporten i samarbejde med disse.
Vi vælger at anvende Unified Process da det er en objektorienteret software-udviklingsproces og derfor kan forenes med Java som er et OOP-sprog[1].

Applikationen vil vi udvikle i programmeringssproget Java med forbindelse til en MySQL Database. Softwarearkitekturen vil vi designe efter de mønstre vi har gennemgået i undervisningen.

Vores endelige produkt skal være en færdig rapport og en fungerende Java-applikation, hvori vi har benyttet ovenstående metoder.

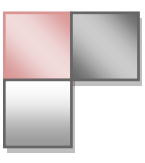
Afgrænsning

af Daniel

Vi vil ved besvarelsen af ovenstående problemstilling begrænse os til at fokusere på produktet som et enkeltbrugersystem, selvom et kursusadministrationssystem er oplagt for flerbrugerproblematikken. Vi vil forsøge at inddrage så meget teori fra undervisningen som muligt, men vi vil begrænse os til det vi finder relevant for opgaven.
Vi har valgt at udelade UP-faserne[2], Construction og Transition. Dette er begrundet ved at Elaboration er den fase hvori de centrale dele i systemet bliver udviklet, og dette er vores mål.
Transition fasen ville aldrig kunne realiseres med den type af system vi udvikler da det som førnævnt lægger op til et flerbrugersystem og at man med stor sandsynlighed ville udvikle det som en webapplikation.

[1] Objektorienteret programmering

[2] Unified Process – Inception, Elaboration, Construction, Transition



Projektetablering

af Gruppen

Rammer

Baggrund

Vores baggrund er at udvikle et enkeltbrugers kursusadministrationssystem.

Opgaven og formål

Opgaven består i at designe og udvikle et nyt kursusadministrationssystem til BEC . Systemet skal være et enkeltbrugersystem der b.la. skal indeholde følgende elementer:

- Kursustilmelding
- Kursusafholdelse
- Evaluering
- Fakturering
- Statistikker

Vores system skal kunne håndtere ovenstående, men uden at kunne håndtere flerbrugerproblematikken. Systemet bliver skabt som en Java-applikation der kører på en PC.

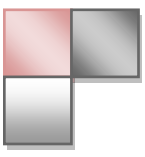
Økonomiske og tekniske rammer

Da dette er et fiktivt skoleprojekt, er der ingen umiddelbare økonomiske rammer for projektet. Vores tekniske rammer er de faciliteter stillet til rådighed af Erhversakademi Sjælland og de private computere vi hver især bidrager med. Udarbejdelse af projektet finder sted primært på Erhversakademi Sjælland.

Kritiske faktorer

Risiko	Konsekvens	Sandsynlig-hed	Omkostning	Total prioritet	Forebyggelse	Løsning
Sygdom	Færre ressourcer, tab af viden	3	4	12	Lev sundt	To om alt.
Kodeviden	Tidskrævende, ringere kvalitet	4	4	16	Par-programmering	Teknisk gennemgang, jævnligt søge viden på nettet/bøger.
Datatab	Tidskrævende, tab af viden	3	5	15	SubVersion, Wiki-side backup	Gem alt minimum to steder.
Børn syge	Færre ressourcer	3	4	12	Sørge for barnepasser ex. bedsteforældre	To om alt vidensdeling. 1 m. børn + 1 u. børn

Figur 1.



Organisering

Projektets organisering

Da gruppens medlemmer er selvvalgte har vi valgt at uddelegere projektets opgaver og ansvarsområder ligeligt.

Ressourcer

Vores viden, vores computere og en virtuel server.

Interessenter

Kunden, BEC, har bestilt et kursusadministrationssystem. I organisationen BEC, er de involverede interessenter:

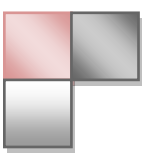
- Kursusadministrator
- Bogholderi
- Kursusansvarlige

Udenfor selve organisationen er der kursusdeltagerne fra BECs kunder.
Interne interessenter: Lars Christian Kofod og Henrik Kryger Høltzer.

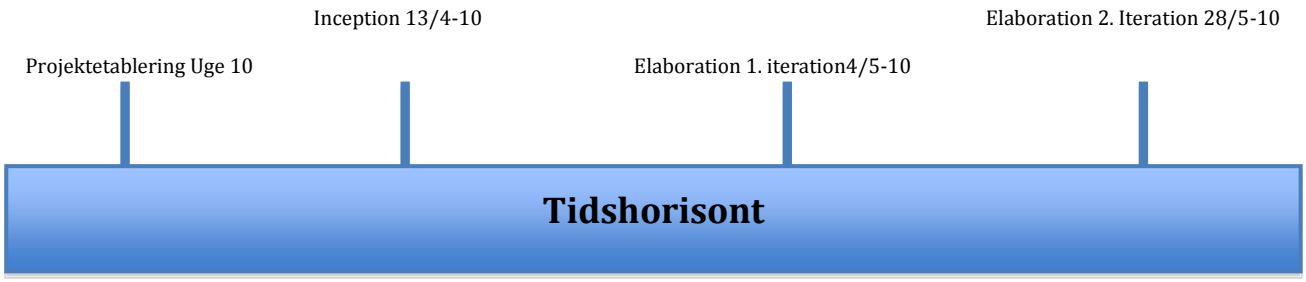
Aftaler og koordinering

Vi arbejder i de skemalagte timer frem til 30. april, derefter arbejder vi mandag-fredag frem til deadline d. 28. maj 2010.

Hjemmeopgaver og eventuelt gruppearbejde i weekend bliver koordineret efter aftale.



Plan



Figur 2.

Milestone – Projektetablering

Under projektetableringen udarbejder vi en gruppekontrakt, identificerer forbundne risici med arbejdsprocessen med en risikoliste. Faserne og de tilhørende iterationer planlægges med deadlines.

Milestone – Inceptionfasen

Virksomheden skal analyseres og derefter skal der udarbejdes en business model. De grundlæggende krav skal identificeres. Vi skal identificere de primære use cases for nøglefunktionerne i systemet og designe en domain model samt programmere en prototype af en grafisk brugergrænseflade.

Milestone – Elaborationfasens 1. Iteration

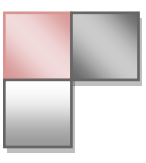
Use case model skal udarbejdes til næsten fuldendt. Systemets arkitektur skal udspecificeres i detaljer. Udarbejdelsen af domænemodel påbegyndes.
Der skal være en prototype af en eksekverbar applikation.

Milestone – Elaborationfasens 2. Iteration

Domænemodellen skal være fuldendt. Design modellen skal være færdig, herunder Design Class Diagram og sekvensdiagrammer. En fuldendt prototype af systemet der kan agere med systemets database.

Arbejdsform

Arbejdsopgaverne i projektet bliver ligeligt fordelt mellem gruppens medlemmer. Vi vil stræbe efter at alle i gruppen får rørt ved alle aspekter i projektets forløb.



Forretningsanalyse

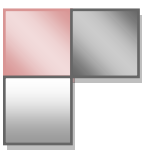
SWOT

Af Stine

SWOT-analysen er en situationsanalyse, ved hjælp af hvilken virksomhedens interne og eksterne forhold kan beskrives, analyseres og vurderes.
Under interne forhold stilles virksomhedens stærke sider (**Styrker**) over for virksomhedens svage sider (**Svagheder**). Her vurderes virksomhedens tekniske, økonomiske og menneskelige ressourcer.
Under eksterne forhold vurderes muligheder (**Muligheder**) og trusler (**Trusler**) i virksomhedens omverden. Her vurderes virksomhedens kunder, konkurrenter, leverandører, finansielle forhold og samfundsforhold.

SWOT analyse for BEC	
Interne forhold	
Styrker	**Svagheder**
• Stor markedsandel af mindre finansvirksomheder. • Veletableret virksomhed med over 40 års erfaring. • "Andelsvirksomhed" • Nyt domicil i Roskilde. • God personalepolitik. • Med opkøbet af ALOC A/S og Schantz Data A/S har de nu et større udbud af services. • Med BEC´s samarbejde med SDC og deres fælles ejede Nordisk Finans IT A/S, sidder de nu tungere på markedsandelen, og har bedre mulighed for at kaste penge i større it-løsninger.	• Svært ved at udvide markedsandele i udlandet, bl.a. på grund af sprogbarrieren og de danske bankregler. • BEC har specialiseret sig i banksektoren, hvor der, accelereret af finanskrisen, nok vil være en reduktion af aktører i den nærmeste fremtid. • Der kommer ikke så mange nye kunder til, da det er svært at starte en ny bank op, samt eksisterende kunder (større banker) har deres egne systemer/egne udviklingsfirmaer.
Eksterne forhold	
Muligheder	**Trusler**
• At bryde den interne sprogbarriere og de udvide til internationale bankregler, og udvide markedsandelen på internationale finansvirksomheder. • Udvikling af bedre sikkerhedssystemer.	• Finanskrisen der har påvirket kunderne og dermed virksomheden. • Lovgivninger. • Renten. • Fusioner banker imellem.

Figur 3.



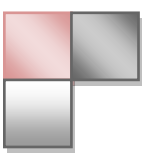
PEST

af Stine

Analysemodellen PEST kortlægger eksterne faktorer som virksomheden ikke selv har indvirkning på. Dens hovedpunkter er politiske, økonomiske (samfundsøkonomiske), sociale og teknologiske forhold for virksomhedens strategiudvikling.
Vi har valgt at benytte PEST-modellen, da dette vil give os et indblik i de faktorer, som påvirker BEC udefra.
Herved anser vi denne model for værende relevant for vores projekt, da det globale IT-marked påvirkes af flere aktører. Eksempelvis staten, der tager politiske beslutninger i form af love og regelsæt, som påvirker BEC indirekte.
Hertil kan vi bruge PEST-modellen til at fremhæve, hvorvidt de eksterne forhold kan have haft en indvirkning på BEC i den givne periode.

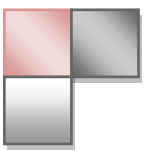
Der er dog et par kritikpunkter af PEST, eksempelvis kan analysen være overfladisk, da denne kan tilpasses enhver branche.
Et andet væsentligt punkt er, at den kan være subjektiv. Dette skyldes at de faktorer og forhold, som vil indgå i analysen, udelukkende bestemmes af brugeren af modellen.

BEC har specialiseret sig på et meget lille område inden for IT, det gør dem også meget følsomme når der så sker noget i den finansielle verden. Eksempelvis har finanskrisen sat tydelige spor hos BEC, som har været tvunget til at starte 2009 med fyringer og nedskæringer for at holde sig på banen, ydermere har BEC startet et samarbejde med SDC for at kunne skære ned på udgifterne til udviklingsopgaver.



PEST analyse for BEC	
Politisk	**Økonomisk**
• Handelshindringer. • Regeringen har lavet en "bankpakke" for at hjælpe banker bedre ud af finanskrisen. • Inflation, renter, kursstigninger og –fald har indflydelse på BEC, som er afhængig af banksektoren.	• De økonomiske faktorer kan have indflydelse på virksomheden, i form af at alle kunder er mindre finansvirksomheder som kan være påvirket kraftigt af f.eks. finanskriser. • Nationalbanker, udvikling i renteniveau • Finanskrisen har påvirket BEC direkte ved eks deres tidligere kunde Roskilde Bank er gået konkurs. Roskilde Bank var en af deres større kunder. • Inflation, renter, kursstigninger og –fald har indflydelse på BEC, som er afhængig af banksektoren.
Socialt	**Teknologisk**
• Finanskrisen har forårsaget en del fyringer i banksektoren og generelt er det mærket på hele arbejdsmarkedet med et ansættelsesstop. • Hackere bliver dygtigere og frækkere, så netbank-systemerne skal sikres bedre. • Virksomheden er afhængig af forbrugernes medievaner, bl.a. deres netbank-systemer. PC´en er blevet alm. i det danske hjem. Selv den ældre del af befolkningen har fået computeren inden for døren.	• Accelererende teknologisk udvikling stiller krav til BEC´s forandringsvillighed. • BEC er en IT-virksomhed, så de er afhængige af at kunderne har hardware til at kunne understøtte deres produkter. • Virksomheden skal altid være et skridt foran mht. udviklingen af software og dens sikkerhed. Konsekvensen af ikke at følge med i udviklingen, kunne være at en konkurrent ville kunne byde ind med en nyere og forbedret teknologi. • Stigende kontrol med den teknologiske udvikling - behov for at beskytte forbrugere og samfundet mod den hastige udvikling i komplekse (og skadelige!) produkter.

Figur 4.



Porters 5 Forces

af Marianne

BEC har 2 primære produkter, nemlig It-løsninger til Pengeinstitutter og It-løsninger til Pensionskasser. Til bankerne udbydes i dag ca. 100 systemer som er udviklet af BEC i samarbejde med bankerne. Datterselskabet Schantz Data A/S har udviklet programmet GALOP (Generelt Administrationssystem Liv Og Pension) som er et system der optimerer aspekterne i pensionsadministration.

Threat of new entrants:

It-løsninger til Pengeinstitutter: Der er ikke umiddelbart nogen trusler. Det er et meget svært marked at komme ind på. Ville kræve kæmpe investeringer og udenlandske firmaer ville få problemer da alt skal foregå på dansk. Både Danske Bank og Nordea har deres egne systemer, men det er systemer som de selv benytter og som ikke udbydes til udenforstående.

It-løsninger til Pensionskasser: Programmet GALOP som BEC Pension tilbyder, er udviklet over flere år og udgifterne til udvikling og drift er delt mellem BEC's kunder. Nye "spillere" på markedet ville have meget store udgifter til udvikling af et nyt produkt og det ville ende med at blive for dyrt for eventuelle kunder.

Bargaining power of customers:

For både Pengeinstitut- og pensionsdelen gælder at kunderne ikke har særlig høj forhandlingsmagt overfor BEC. Dette skyldes at der ikke umiddelbart er nogen alternativer og at det ville have meget store omkostninger for kunderne at udvikle egne systemer.

Bargaining power of suppliers:

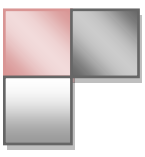
De forskellige leverandører til BEC har ikke nogen særlige forhandlingsmagt. Leverandører af hardware og software vil nemt, og uden større omkostninger for BEC, kunne udskiftes. Andre leverandører vil være leverandører af el, vand, renovation og lignende og disse har heller ingen forhandlingsmagt.

Threat of substitutes:

Der findes ikke umiddelbart nogen substituerende produkter for hverken pengeinstitut- eller pensionsdelen.

Degree of rivalry

Da der ikke er nogen direkte konkurrenter er der heller ikke nogen høj grad af rivalisering.



Boston Matrix

Af Martin

Boston Matrix info

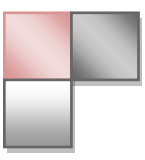
Boston Matrix er et business-analyseringsværktøj, vi bruger til at analysere et firmas egne produkter op imod andre produkter på samme marked, som de skal konkurrere imod. Det kan være brugbart til at sortere produkt- eller produktlinjer man vil analysere. Man vil derfor opnå info omkring hvilke produkter der giver overskud, mulighed for overskud eller produkter/idéer der skal skrottes. Boston Matrixen er et kvadrat opdelt i fire kvadratiske felter, der hver repræsenterer fire forskellige tilstande et produkt kan have. Man ser matrixen som et koordinatsystem hvor x-aksen repræsenterer hvor hurtigt markedet vokser og y-aksen hvor stor markedsandelen er.

Det eneste minus ved denne model er at den på nogle områder godt kan afbillede situationen som værende dårlig forretning selvom den er indbringende for firmaet. F.eks. kan et produkt der ligger i kategorien **dog** godt være indbringende, selvom den på modellen ser ud til at være spild af penge. Et firma kan godt have et produkt med lav markedsandel på et marked der ikke er i vækst men samtidig lave overskud på produktet. Et eksempel kunne være styresystemet OS X. Apple har en lav markedsandel, på et marked hvor der kun er to kommercielle spillere, Microsoft og Apple. Men sandheden er, at Apple hvert år har overskud på produktet OS X, men på en Boston Matrix ville den være dømt ude.

Dog
De produkter der ligger i dette kvadrant har en lav markedsandel samt ingen fremtid for udvidelse af markedet. Produkter der ligger i dette kvadrant er firmaets, økonomisk set, værste produkter. Et eksempel kunne være et produkt som koffeinfri kaffe fra en vilkårlig dansk kaffeproducent. Koffeinfri kaffe er ikke et særligt benyttet produkt indenfor kaffeverdenen i Danmark, men de fleste kaffeproducenter producerer det alligevel. De producerer for ikke at tabe markedsandelen, selvom den stort set er ikke-eksisterende, til de andre producenter. Selve markedet for koffeinfri kaffe slår nok aldrig igennem i Danmark, som det er set i USA.

Cash cow
Produkterne i denne kategori har en stor markedsandel på et ikke-voksende marked. Firmaer hvis produkter ligger i denne kategori får valuta for pengene, da de ikke behøver at markedsføre deres produkter, da det har en kæmpe andel, jo længere vi kommer ud ad y-aksen. Et eksempel kunne være Coca-Cola Companys Cola, der er verdenskendt og kan købes i stort set alle lande i verden. Alle kender den og den har en kæmpe markedsandel og behøver ingen markedsføring. Alligevel bruger Coca Cola company hvert år svimlende summer på markedsføring, da de ikke vil væltes af pinden. Produkterne i denne kategori er derfor gode at beholde, da de kaster masser af penge af sig.

?
Spørgsmålstegnet er produkter som har en lille markedsandel på et hurtigt voksende marked. Det er altid godt for et firma at være med på et marked der vokser hurtigt, men det gælder også om at have en betydelig markedsandel, for ellers bruger man mange penge der aldrig kommer til at tjene sig ind



igen, lige modsat Cash cow. Det kan derfor være en fordel at lægge en marketing-strategi og se om det virkelig kan betale sig at være med i "kampen" eller om man skal nedlægge produktet.

Er det realistisk at ens produkt slår igennem og får en højere markedsandel? Et eksempel kunne være Motorolas mobiltelefoner, der konkurrerer på et ekstremt hurtigt voksende marked, men stort set ingen andel har i Danmark.

Stars
Produkterne i denne kategori er de bedste produkter ens firma kan have. De er i et marked der er hurtigvoksende og har en stor andel. Hvis et firma har produkter liggende i denne kategori, er det godt at blive ved med at investere i, da det vil bære frugt og give flere penge igen, end hvad man har brugt på produktet. Det kan ende med at man i fremtiden sidder med hele markedsandelen. Et eksempel kunne være styresystemet Windows, der op igennem tiden har kæmpet hårdt på et hurtigt voksende marked, der i dag har opnået næsten total markedsandel indenfor styresystemer på klientsiden.

BECs produkter

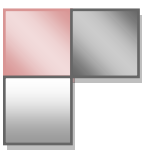
Virksomheden BEC tilbyder mange forskellige elementer indenfor finansområdet. På bec.dk kan man finde information omkring de produkter de tilbyder deres kunder indenfor banksektoren.

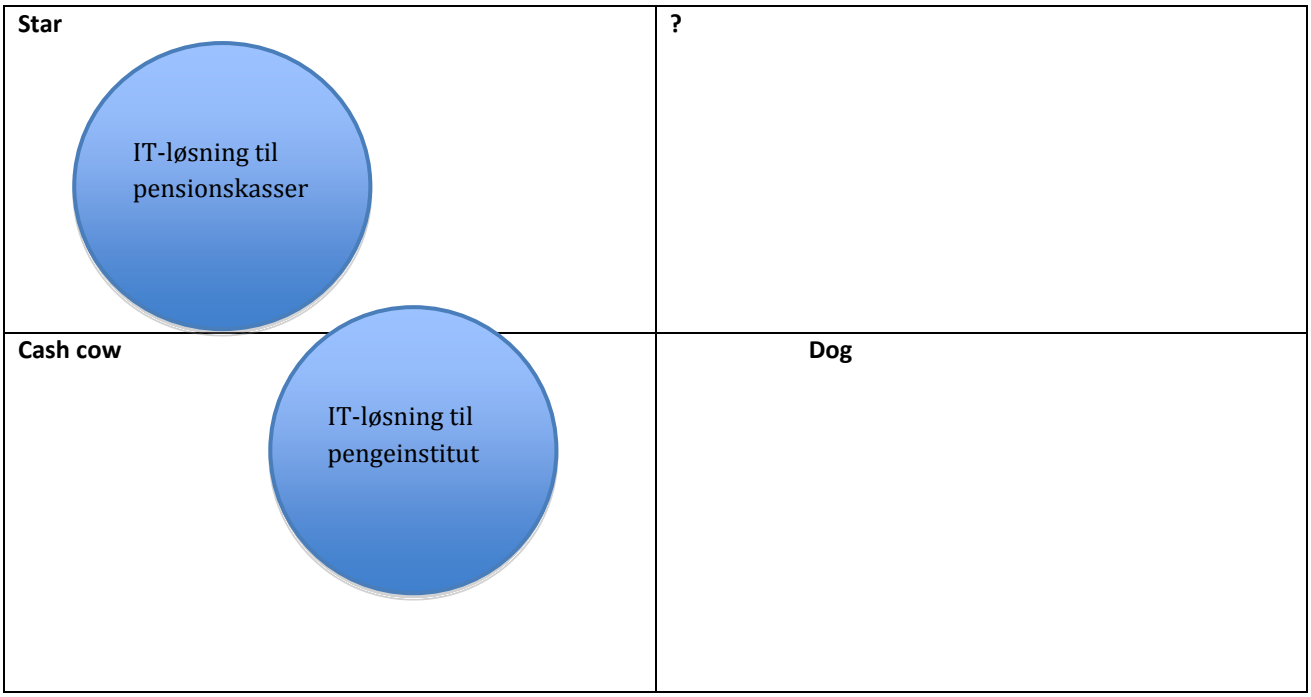
Løsninger til pengeinstitutter

- Kunde- og kontoadministration samt betalingskort
- Dansk og udenlands betalingsformidling
- Selvbetjening: Netbank, sms, e-mail, telefon og pengeautomat
- Handel, depot og formueforvaltning
- Rådgivning: Pension, lån/kredit, investering, formue, bolig og daglig økonomi
- Salg, markedsføring og relationer
- Kredit- og risikostyring
- Regnskab og myndighedsrapportering
- Ledelsesinformation
- Total arbejdspladsstyring inkl. IP-telefoni

Kilde: bec.dk

Vi har samlet deres produkter i en boks og kaldt dem "Løsninger til pengeinstitut" og placeret dem i vores Boston Matrix model. Vi har analyseret produkterne BEC tilbyder og analyseret det marked hvor BEC laver forretning. Vi har ikke kunne finde nogen direkte konkurrenter til BEC, da de tilbyder en unik all-in-one pakkeløsning til mindre banker og pensionskasser. Der findes PBS, men de udvikler næsten kun betalingskort. Store banker såsom Nordea, Danske Bank og Jyske Bank, udvikler og har deres helt egen IT-afdeling, så for dem ville det ikke kunne betale sig at samarbejde med BEC. Men for de mindre banker og pensionskasser er BEC den optimale løsning, da de ikke skal bruge penge på personale og udvikling.





Figur 5.

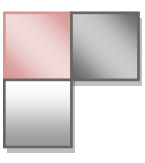
Vi har samlet BECs produkter i to kategorier og kaldt dem henholdsvis:

- IT-løsning til pensionskasser
- IT-løsning til pengeinstitut

Vi har valgt at placere BECs produkt, IT-løsning til pensionskasser, i kategorien "Stars", da BECs unikke forretningsmodel er indbringende for BEC. Vi har ikke placeret produktet helt ude til venstre, da BEC ikke har al markedsandel, da de store banker som sagt har deres eget, der måske nok er større end alle BECs kunder tilsammen.

Det andet produkt, IT-løsninger for pengeinstitutter, har vi placeret i kategorien "Cash cow", da BEC stadig har en relativ stor markedsandel, da de tilbyder et unikt produkt for mindre banker. Det kan være en fordel for små banker, da systemer bliver billigere jo flere kunder BEC har, da bankerne deler regningen for systemet.

Markedet hvor BECs produkter konkurrerer i, kan ikke rigtig vokse sig større, kun hvis de får kunder som Nordea og Danske Bank. BEC er en national firma og har udelukkende kun danske kunder og kan derfor ikke udvide internationalt. Det er også derfor BEC kan tilbyde unikke produkter, da deres systemer og support udelukkende fungerer på dansk. Det kan være svært for internationale virksomheder med samme domæne som BEC at få danske kunder da prisen umiddelbart ville være højere end den BEC kan tilbyde.

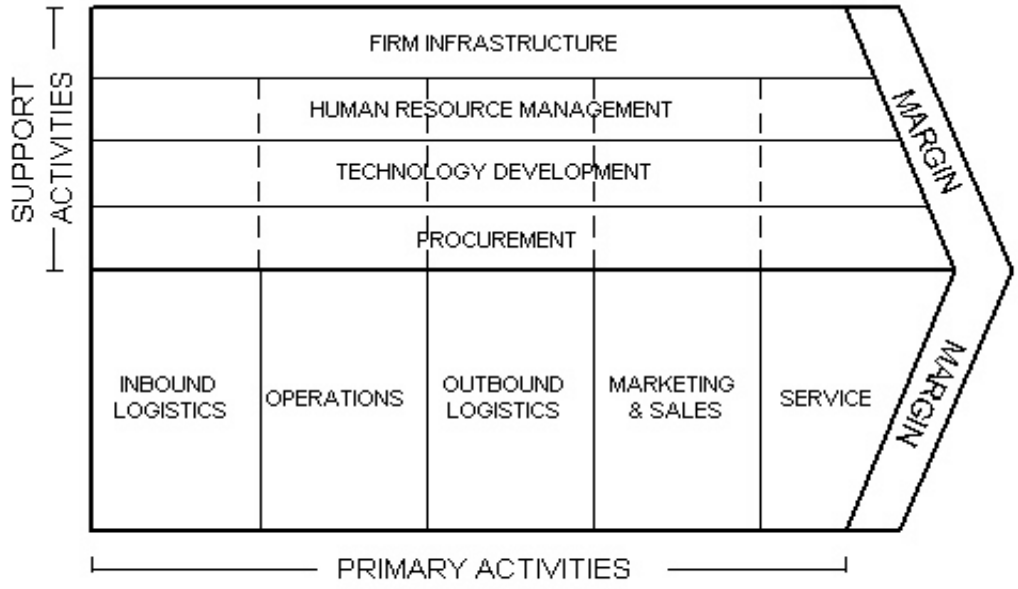


Porters Værdikæde

af Stine

Porters værdikæde beskriver hvor og i hvilke aktiviteter virksomhedens produkter får en merværdi. Denne analysemodel er dog skabt til produktionsvirksomheder, men kan bruges på servicevirksomheder.
Gennem de forskellige faser i modellen skabes der er en merværdi for den færdige vare, men i realiteten kan hver fase igen deles op i mere specifikke aktiviteter alt afhængig af hvilken branche der arbejdes med.

Nedenunder ses værdikædemodellens værdiaktiviteter, der er de byggesten virksomheden benytter for at skabe et produkt. Værdikædens margin symboliserer forskellen mellem den totale værdi og den omkostningsmæssige udgift ved at udfører disse værdiaktiviteter, altså fortjenesten.



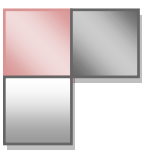
The Generic Value Chain
Kilde: Porter: Competitive Advantage. 1998.

Figur 6.

Support

Administration, ledelse og infrastruktur:

Ledelse og administration: BEC er ejet af 55 medlemmer, der alle er mindre pengeinstitutter fordelt i Danmark. Derudover tæller BEC´s kundekreds over 200 finansielle virksomheder, der er spredt over hele Danmark, Færøerne og Grønland.
BEC har 4 datterselskaber; Schantz Data A/S, BEC Pension A/S, ALOC A/S, BEC Ejendomsselskab A/S.
I udgangen af 2009 havde BEC 901 ansatte og i hele koncernen var der 1050 fuldtidsansatte.



Ledelsen i BEC består af en bestyrelse på fem medlemmer, som hver især har en direktørstilling repræsenteret fra hver deres medlemsbank. I selve direktionen sidder Leo Svendsen som Adm. Direktør og som direktør sidder Kurt Nørrisgaard. Derunder kommer så alle de forskellige afdelingers ledere.

BEC's fremtidige **planlægning** er blevet forpurret lidt af den nuværende finanskrise, men som de selv udtrykker sig vil de komme styrket ud på den anden side.
2010 vil blive præget af opbygningen og realiseringen af deres nye strategiske samarbejder med SDC om JN Data.
BEC´ s nye flagskib "GALOP" inden for pensionsområdet satser de meget på og i slutningen af 2010 vil også JØP(Juristernes og Økonomernes Pensionskasse) benytte "GALOP".

Infrastruktur: BEC startede i 2009 med at modernisere deres netværk, så deres kapacitet nemt kan udbygges både på deres eget netværk men også på kundeforbindelserne. Det skal sikre at nedbrud ikke bevirker en reduceret tilgængelighed for kunderne.
I BEC Direkte og i System Support er ventetiden gennemsnitlig på ca.1 minut, som ligger langt under deres mål på 2 minutter. Hos Netbank Support besvarer BEC ca. 70.000 opkald og 4000 e-mails hvert år. Ventetiden på disse opkald er lige knap 2 minutter og ligger igen under deres planlagte max ventetid på 5 minutter. Virksomheden sørger selv for nødstrøm og køl til deres servere.
BEC har baseret deres it-sikkerhed på en godkendt dansk standard og udover dette har de implementeret en sms-service som bruges som verificering af udvalgte transaktioner.

Personaleudvikling og -styring:

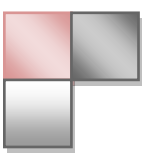
BEC lægger meget stor vægt på ansattes trivsel og sundhed, dette gør de med bl.a. sund mad og mulighed for fysisk udfoldelse. Virksomheden bruger mange ressourcer på at deres ansatte trives. Uddannelse vægtes højt i BEC, faktisk kræver de at deres ansatte følger med i udviklingen. I 2008 brugte virksomheden 14 mio. kr. på videreuddannelse. Alle ledere skal deltage i et større BEC-udviklingsforløb og et individuelt udviklingsforløb inden for lederkommunikation.
Udover dette har BEC udarbejdet et beredskab i tilfælde af en epidemi af H1N1, eksempelvis har de øget muligheden for at arbejde hjemmefra.

Teknologiudvikling:

Den IP-telefoni BEC udbyder har de tidligere selv stået for, men deres platform er nu så forældet at de har valgt at udlicitere det til TDC, da en opgradering ville koste temmelig meget.
I løbet af 2008 gik BEC i gang med at udskifte deres alm. servere med virtuelle servere så de på den måde vil spare ca. 80 %.
BEC er en IT-virksomhed så de lever jo af at udvikle og videreudvikle it-løsninger.

Indkøb:

El fra SEAS-NVE+ alm. omkostninger til virksomhedens bygninger og andre alm. virksomhedsomkostninger(papir, rengøring, mm). Platformen "Calypso", der er specialiseret i løsninger til kapital-markedet, er til dato være den største købeløsning for BEC.



Produktionsanlæg, maskiner, driftsmateriel og inventar står på BEC´s årsregnskab, men vi kan ikke nogen steder i deres årsrapport finde nogen udspecificering på dette.

Primære aktiviteter

Indgående logistik:

BEC råder ikke over noget lager som sådan, da deres produkt er software. Men de benytter sig dog af "færdiglavede" produkter til videreudvikling og videresalg. De har dog deres servere hvor alle deres produkter og løsninger er på. Megen kommunikation imellem afdelingerne foregår også igennem e-mails eller andre it-baserede løsninger.

Produktion:

De investerer hvert år et 3 cifret antal mio. af kroner til udvikling og testning af deres it-løsninger/produkter. Første skridt på vejen til det færdige produkt er projektgruppen, der udarbejder en løsningsmodel til projektet/systemet. Derefter ryger det videre til systemudviklerne der er bindeleddet mellem forretningen og den rå kode. Det 3. skridt er selve koden og sidste skridt i udviklingsprocessen er test af selve det færdige projekt/system.

Udgående logistik:

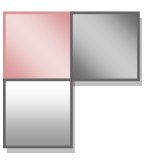
BEC udvikler deres produkter sammen med deres kunder, så disse er med i hele udviklingsprocessen. Igen har BEC ikke brug for nogen større lagerplads. Når tiden er til at implementere det færdige produkt, sørger BEC for den fulde installation hos de pågældende kunder.

Markedsføring og salg:

BEC har en fast kerne af kunder(medlemmer)og det er naturligt at når der er behov for nyt software benytter disse sig af BEC. Udover dette "partnerskab" sælger BEC sig selv på nettet på deres ganske udmærkede hjemmeside, der både henvender sig til fremtidige kunder men også til mulige ansatte i fremtiden.

Service:

BEC er det, de selv kalder et "full service it-hus". De har support til både deres kunder direkte, men også til deres kunders kunder. Som nævnt tidligere kan de, alt afhængig af individuel aftale, udvikle helt fra bunden og fortsætte med at servicere deres kunder, til disse ikke er kunder længere.



Forretningsmodel

Af Marianne

Marked:

Markedet består primært af danske pengeinstitutter og pensionskasser. Enkelte udenlandske pengeinstitutter har købt BEC's produkter i forbindelse med åbning af danske filialer. Det har i 2009 været et turbulent marked pga. finanskrisen, men der forventes mere stabilitet og måske endda en smule vækst i 2010.

Tilbyder:

Produkter:

1. IT-løsninger til pensionskasser, herunder

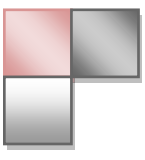
    - Web-grænseflade til sagsbehandler og medlem
    - Kundeadministration, CPR-opdatering, tilbud
    - Hændelsesstyring, indbetaling, udbetaling, indberetning.
    - Fremregning, prognose, frem- og tilbagerulning, versionsstyring og historik
    - Print, standardbreve og arkiv
    - Total arbejdspladsstyring inkl. IP-telefoni.

2. IT-løsninger til pengeinstitutter, herunder

    - Kunde- og kontoadministration samt betalingskort
    - Dansk og udenlandsk betalingsformidling
    - Selvbetjening: netbank, sms, e-mail, telefon og pengeautomat
    - Handel, depot og formueforvaltning
    - Rådgivning: pension, lån/kredit, investering, formue, bolig og daglig økonomi
    - Salg, markedsføring og relationer
    - Kredit- og risikostyring
    - Regnskab og myndighedsrapportering
    - Ledelsesinformation
    - Total arbejdspladsstyring inkl. IP-telefoni.

Organisation:

BEC har overordnet en bestyrelse og en direktion med henholdsvis en bestyrelsesformand og en administrerende direktør i spidsen. Der er 5 afdelinger der refererer til direktionen: Sikkerhed, økonomi, HR, jura og Kommunikation & markedsrelationer. I spidsen af disse sidder en sikkerhedschef, en CFO (Chief Financial Officer), en HR-direktør, en Chefjurist og en kommunikations-



og marketingdirektør. Under disse findes der yderligere 5 underafdelinger med hver deres direktør: Salg og relation, Fundament, Udvikling pengeinstitut, Pension & kompetence og Drift & support.

Aktiviteter:

BEC's primære aktivitet er at udbyde full service it løsninger til pengeinstitutter og pensionskasser. BEC står for både implementering, support og drift. Dette varetages af virksomhedens forskellige afdelinger. BEC har stået for egen IT-drift men indgik i 2009 i et samarbejde med JN Data. Aftalen går ud på at JN Data i løbet af 2010 skal overtage IT-driften af BEC.

Ressourcer:

1. Engagerede og erfarne medarbejdere, gennemsnitlig anciennitet er 8 år, som der hele tiden satses på at kompetenceudvikle og efteruddanne.
2. Pensionsadministrationssystemet GALOP som er udviklet af datterselskabet Schantz Data A/S.

Suppliers:

1. Leverandører af kontorartikler.
2. Leverandører af hardware.
3. Leverandører af software.
4. Leverandør af el og vand.
5. Leverandør af renovation.

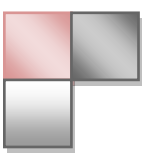
Strategier:

IT:

1. BEC indgik i slutningen af 2009 et samarbejde med Skandinavisk Data Center(SDC) og stiftede hermed selskabet Nordisk Finans IT(NFIT) og dette skal være drivkraften for en fælles IT-strategi.
2. Man har i tæt samarbejde med medlemsbankerne bl.a. satset på udvikling af systemet Calypso, som er et system til håndtering af capital market-produkter i bankerne, fx aktier, obligationer og valuta.

Økonomi:

1. Man har i 2009 satset på at skære i omkostningerne pga. finanskrisen og dette er lykkedes så godt at man har vendt et underskud i 2008 på 49 mio. til et overskud på 1 mio. med næsten identiske omsætninger for de to år. BEC forventer yderligere fremgang i 2010, især pga. samarbejde med SDC og et driftssamarbejde med JN Data som vil overtage driften allerede her i 2010. Desuden satses på salg af systemet GALOP gennem datterselskabet Schantz Data A/S.



HRM:

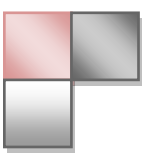
1. Alle nye medarbejdere kommer gennem et særligt introduktionsprogram
2. Der satses på efteruddannelse for alle medarbejdere. I 2008 blev der brugt 14 mio. på efteruddannelse
3. For ledere gælder at de alle har været gennem et stort BEC-udviklingsforløb. Lederne bliver løbende målt via arbejdspladsvurderinger og tilfredshedsundersøgelser blandt medarbejderne.

Konklusion forretningsdel

BEC sidder umiddelbart i en meget gunstig situation. Konkurrence er der ikke meget af og deres største konkurrent, Skandinavisk Data Center er det lykkedes dem at indgå et samarbejde med og de er hermed elimineret som konkurrent. 2009 har været et vanskeligt år rent forretningsmæssigt, men det ser ud til at blive bedre i 2010 og der forventes endda en smule vækst.

Pensionsmarkedet er i vækst og programmet GALOP vil hjælpe med til at få nye kunder også uden for ejerkredsen. På markedet for pengeinstitutter er BEC mange steder ved at implementere deres nye platform Calypso som udvikles sammen med SDC i det nydannede NFIT, så også her ser fremtiden lys ud.

Den fremtidige IT-strategi ligger i den nye samarbejdspartner JN Datas hænder. Formålet med det nye samarbejde er at sikre stordriftsfordele med tilhørende reducerede driftsomkostninger. Derudover vil samarbejdet medføre en mere effektiv it-infrastruktur som skal styrke konkurrencekraften.



Vision

af Daniel og Martin

BEC har efterspurgt et system der letter arbejdsgangen ved at automatisere de administrative funktioner ved

- Kursustilmelding
- Kursusafholdelse
- Fakturering
- Evaluering
- Statistikker

Vi har ud fra det valgt at udvikle et kursusadministrationssystem, der skal være overskueligt og enkelt for en kursusdeltager at tilgå. Derfor vil vi lægge vægt på brugergrænsefladen og måden kurserne bliver præsenteret for den eventuelle deltager på. Derudover skal systemet være nemt at vedligeholde og administrere for den kursusansvarlige og administrationen. Oplysninger skal være nemme at tilgå afhængigt af hvilke rettigheder brugeren har.

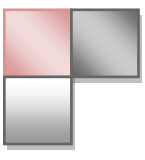
Vedrørende brugergrænsefladen vil vi her især lægge vægt på at det skal være nemt at navigere mellem de forskellige faner. Informationer der vedrører kurser skal være præsenteret på en overskuelig måde. Yderligere skal brugen af musen minimeres så al funktionalitet skal kunne udføres ved brug af tastatur.

Aktørernes hverdag

Klokken er otte og den kursusansvarlige Mette møder på sin arbejdsplads hos BEC. Hendes første opgave i dag er at oprette et nyt kursus i det nye kursusadministrationsssystem. Kurset hun skal oprette hedder "Java for begyndere" og skal afholdes i BEC's egne lokaler i Roskilde.
Mette logger på systemet og vælger fanebladet "Opret Kursus". Hun indtaster alle oplysninger vedr. kurset og tilknytter en af de undervisere der allerede er oprettet i systemet.

Jens fra Morsø Bank føler at han har stået stille på sin arbejdsplads i en tidsperiode og har nu besluttet at søge efteruddannelse for at få nye udfordringer. Han har hørt om BEC's nye kursussystem, og logger for første gang på systemet med det login han har fået tilsendt på sin e-mail. Jens vælger fanebladet "Kurser", hvor han har et træ med en oversigt over alle tilgængelige kurser. Hans øje fanger kurset "Java for begyndere", og skynder sig straks at tilmelde sig da han bemærker at der kun er én ledig plads tilbage. Han tjekker hurtigt sin kalender for at se om han er ledig i afholdelsestiden for kurset og da alt er i orden trykker han på knappen "Tilmeld". Jens kan nu på fanebladet "Egne tilmeldinger" se at han afventer at blive godkendt til kurset "Java for begyndere". Der står at han vil modtage en e-mail når hans tilmelding er godkendt.

Den kursusansvarlige Mette, går på systemet for at tjekke for nye indkomne kursustilmeldinger.
Hun vælger fanebladet Tilmeldinger hvor hun har en oversigt over alle tilmeldinger i et træ.
Hun bemærker at for kurset "Java for begyndere" er der et navn der er highlighted med rød skrift, som nu står klar til godkendelse. Mette klikker på Jens, og i en tabel får hun vist alle data vedrørende Jens.
Hun anerkender at alle data er i orden, og godkender dermed hans tilmelding i systemet.



Systemets aktører og roller

af Martin

I vores kursusadministrationssystem er der forskellige aktører og roller man kan fordele på forskellige brugere af systemet. Vores system skal kunne en del, men langtfra alle funktioner skal være tilgængelige for alle. I vores projektoplæg var der fire roller: Kursusdeltager, Kursuadministrator, Kursusansvarlig og Bogholderiet.

Vi har valgt at ekskludere kursusadministratoren fra denne version af opgaven, da hans arbejdsopgaver omtaler extranettet som i vores øjne ligger op til en webapplikation og derfor er uden for pensum.

Kursusdeltager

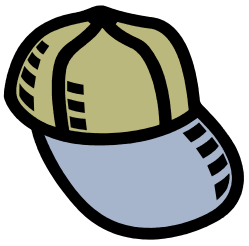
Rollen "kursusdeltager" i vores system er repræsenteret med en grøn kasket. Kursusdeltageren skal kunne få vist en liste over kurser BEC tilbyder til deres kunder. Det skal være muligt for deltageren at tilmelde sig et eller flere af de viste kurser i systemet. Deltageren skal kunne se de kurser han er tilmeldt til og se info ang. kurset. En tilmelding sker ved at deltageren vælger et kursus og trykker tilmeld.

Kursusansvarlig

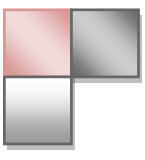
En kursusansvarlig i vores system er personen der administrerer alle kurserne. Hans opgave er at oprette kurser, kursusbeskrivelser, godkende tilmeldinger osv. Den kursusansvarlige kan oprette et kursus i et grafisk interface vi designer, med et kursusnavn- og beskrivelse, sted, dato og tid, altså al praktisk information omkring kurserne. Man bliver godkendt af den kursusansvarlige til et eller flere kurser, da systemet bliver designet således at man ikke direkte er tilmeldt når en deltager tilmelder sig selv.

Fakturering er også et vigtigt element af den kursusansvarlige funktion. Der skal sendes faktureringsoplysninger videre til bogholderiet, så der kan sendes fakturaer ud til de kunder, som de deltagende arbejder hos.

Bogholderiet

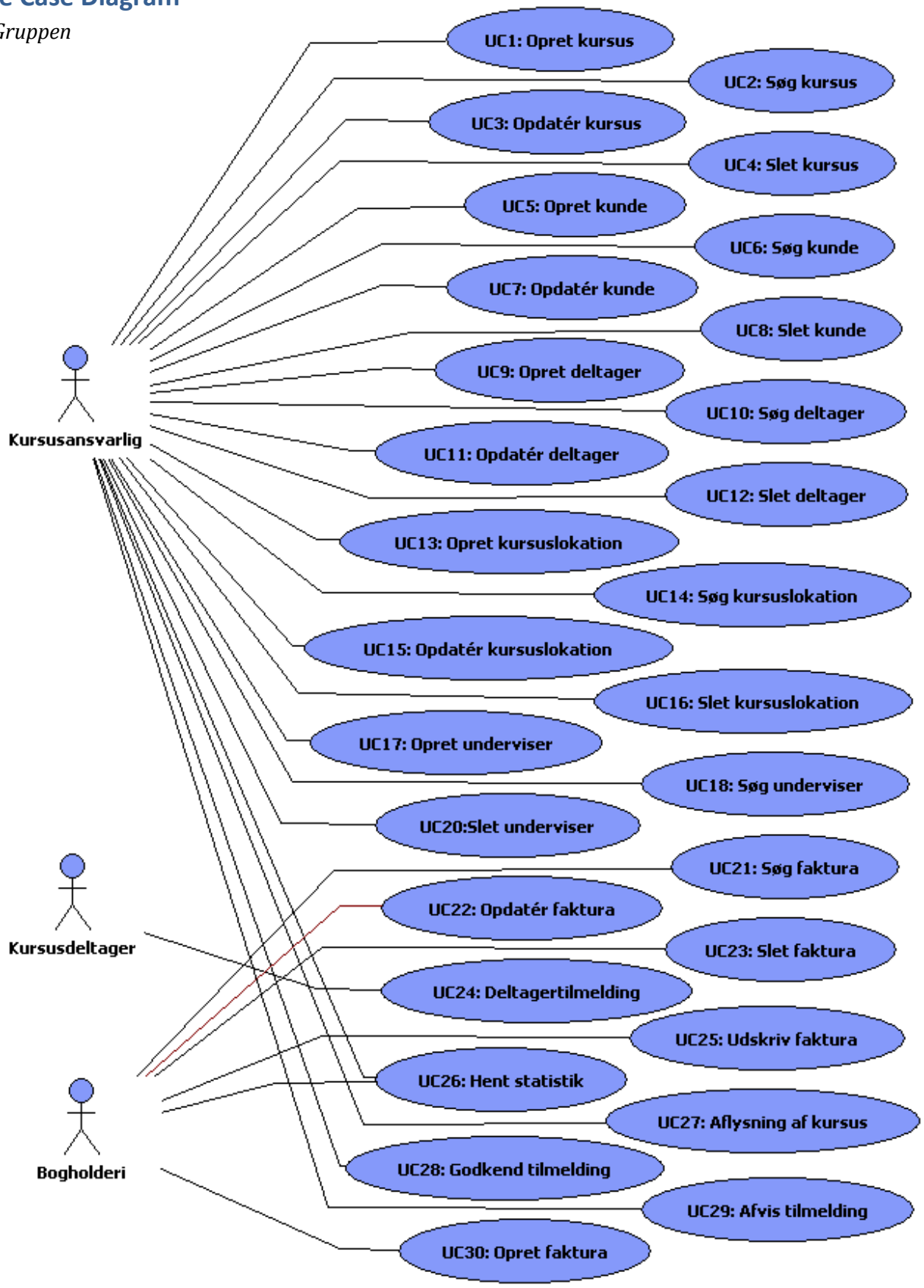


Bogholderiet modtager faktureringsoplysninger fra den kursusansvarlige og sender faktura ud til de respektive kunder, hvis medarbejdere har deltager i kurser hos BEC. De får en faktura hvor der fremgår hvem, hvilket og hvornår kurserne er blevet afholdt, samt en total pris de skal betale.



Use Case Diagram

af Gruppen



Figur 7.

Use Case Text

af Gruppen

UC. 1
Opret kursus
Kursus oprettes med navn, kursusbeskrivelse, lokation og en fastsat start- og slutdato. En underviser kan tilknyttes men er ikke nødvendig for oprettelse.
Primary actor: Kursusadministrator.
Stakeholders & other interests: Bogholderiet.
Pre-condition: Der skal være oprettet lokationer i databasen.
Post-condition: Kurset er oprettet og gemt i databasen.
Main succes scenario:

- Fanebladet "Kurser" vælges.
- Kategori vælges fra dropdown.
- Kursustitel indtastes.
- Kursusbeskrivelse inden for denne kategori vælges.
  a. Tryk "Vis fuld tekst"
  b. Gå videre til datoindtastning.
- Start- og slutdato indtastes.
- Start- og sluttidspunkt indtastes.
- En underviser vælges fra dropdown.
- Tryk "Opret".
- Systemet gemmer kurset i en lokal ArrayList og gemmer derefter i Databasen.

Alternative scenarios:

- Manglende/ulovlige oplysninger udløser fejlmeddelelser.
- Ingen forbindelse til databasen udløser fejlmeddelelse. En dialogboks kommer frem, hvor kursusadministrator kan oprette forbindelsen igen.
- Systemet går ned.

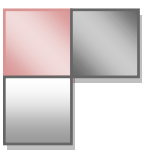
Special requirements:

- Der skal være lave svartider på oprettelse, max. 3 sekunder
- Alle funktioner skal være på dansk

UC. 2
Søg kursus
Søgning på oprettede kurser. Der kan søges på de forskellige kriterier; underviser, kursusnavn, kursusbeskrivelse og datoer.

UC. 3
Opdater kursus
De i UC. 1 nævnte parametre kan ændres.

UC. 4
Slet kursus
Kursus slettes fra databasen



UC. 5
Opret kunde
Kunde oprettes med firmanavn, CVR-nr, adresse, kontooplysninger og kontaktperson.

UC. 6
Søg kunde
Søgning på oprettede kunder. Der kan søges på de forskellige kriterier; firmanavn, CVR-nr, adresse og konooplysninger.

UC. 7
Opdater kunde
De i UC. 5 nævnte parametre kan ændres.

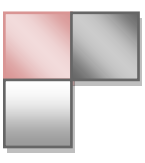
UC. 8
Slet kunde
Kunde slettes (flyttes i andet kartotek)

UC. 9
Opret deltager
Deltager oprettes med firmanavn, fornavn, efternavn, adresse, postnummer, e-mail og telefonnummer.

UC. 10
Søg deltager
Søgning på deltager. Der kan søges på de forskellige kriterier; firmanavn, fornavn, efternavn, adresse, postnummer, e-mail og telefonnummer.

UC. 11
Opdater deltager
De i UC. 9 nævnte parametre kan ændres.

UC. 12
Slet deltager
Deltager slettes fra databasen.



UC. 13
Opret kursuslokation
Kursuslokation oprettes med stednavn, postnummer og adresse.
Primary actor: Kursusansvarlig(Kansv)
Pre-condition: By med tilhørende postnummer skal være oprettet I databasen.
Post-condition: En kursuslokation er oprettet og gemt i databasen.
Main succes scenario:

- Aktøren vælger fanebladet "Opret Kursuslokation".
- Der indtastes stednavn, postnummer og adresse.
- Aktøren trykker på knappen "Opret".

Alternative scenarios:

- Manglende/ulovlige oplysninger udløser fejlmeddelelser. Kansv taster de manglende/retter de ulovligeoplysninger, og trykker på "opret kursus"
- Ingen forbindelse til databasen udløser fejlmeddelelse. En dialogboks kommer frem, hvor kansv kan vælge at oprette forbindelsen igen.
- Systemet går ned. Kansv genstarter systemet og checker om hendes kursus er blevet oprettet ellers starter hun forfra.

Specielle krav:

- Der skal være lave svartider på opret, max. 3 sekunder
- Alle funktioner skal være på dansk

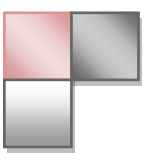
UC. 14
Søg kursuslokation
Søgning på kursuslokation. Der kan søges på de forskellige kriterier; stednavn, postnummer og adresse.

UC. 15
Opdater kursuslokation
De i UC. 13 nævnte parametre kan ændres.
UC. 16
Slet kursuslokation
Kursuslokation slettes fra databasen.

UC. 17
Opret underviser
Underviser oprettes med fornavn, efternavn, postnummer, adresse, e-mail, og telefonnummer.

UC. 18
Søg underviser
Søgning på underviser. Der kan søges på de forskellige kriterier; fornavn, efternavn, postnummer, adresse, e-mail og/eller telefonnummer.

UC. 19
Opdater underviser
De i UC. 17 nævnte parametre kan ændres.



UC. 20
Slet underviser
Underviser slettes fra databasen.

UC. 21
Søg faktura
Søgning på faktura. Der kan søges på de forskellige kriterier; firmanavn, fakturanummer og deltagernavne.

UC. 22
Opdater faktura
De i UC. 21 nævnte parametre kan ændres.

UC. 23
Slet faktura
Faktura slettes i databasen.

UC. 24
Deltagertilmelding til kursus
Deltageren tilmeldiger sig et kursus.
Primary actor: Kursusdeltager
Stakeholders & other interests: Kursusansvarlig
Pre-condition: Der skal være oprettet et eller flere kurser i systemet og deltageren er logget ind.
Post-condition: Deltageren har sendt en eller flere tilmeldinger.
Main succes scenario:

- Deltageren vælger et kursus i træet.
- Systemet viser data for det givne kursus.
- Kursusdeltageren trykker på ”Tilmeld”.
- Systemet returnerer en tråd om at tilmeldingen er sendt til godkendelse.

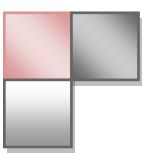
Alternative scenarios:

- Systemet går ned.
- Deltageren udfører en handling der ikke opfylder en forretningsbetingelse (Se afsnit om domæneregler).

Special requirements:

- Der skal være lave svartider, max. 3 sekunder
- Alle funktioner skal være på dansk

UC. 25
Udskriv faktura
Faktura findes ved hjælp af fakturanr. og udskrives.



UC. 26
Hent statistik
Der trækkes statistik ud fra de ønskede kriterier og disse udskrives.

UC. 27
Aflysning af kursus
Der findes det kursus frem der skal aflyses. Kurset slettes og alle tilmeldte deltagere får besked via deres e-mail.

UC. 28
Godkend tilmelding
Den kursusansvarlige godkender deltagertilmeldingen hvis alle kriterier er opfyldt.
Primary actor: Kursusansvarlig(Kansv)
Stakeholders & other interests: Bogholderiet og deltager.
Pre-condition: En deltager har medlt sig til et kursus.
Post-condition: Kansv har godkendt tilmeldingen, ændringen gemmes i databasen og lokalt i en ArrayList.
Main succes scenario:

- Kansv vælger fanebladet ”Tilmeldinger”
- Kansv vælger den deltagertilmelding der skal godkendes
- Kansv trykker på ”godkend”
- Den røde skrift ved siden af navnet ændres til et grønt ”godkendt”
- Ændringen gemmes i databasen og lokalt i ArrayList

Alternative scenarios:

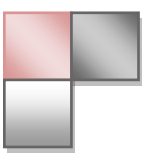
- Ingen forbindelse til databasen udløser fejlmeddelelse. En dialogboks kommer frem, hvor kursusadministrator kan oprette forbindelsen igen.
- Systemet går ned. Kursusadministrator genstarter systemet og checker om godkendelsen er gemt ellers starter hun forfra.

Specielle krav:

- Der skal være lave svartider, max. 3 sekunder
- Alle funktioner skal være på dansk

UC. 29
Afvis tilmelding
Den kursusansvarlige afviser deltagertilmeldingen hvis alle kriterier ikke er opfyldt.

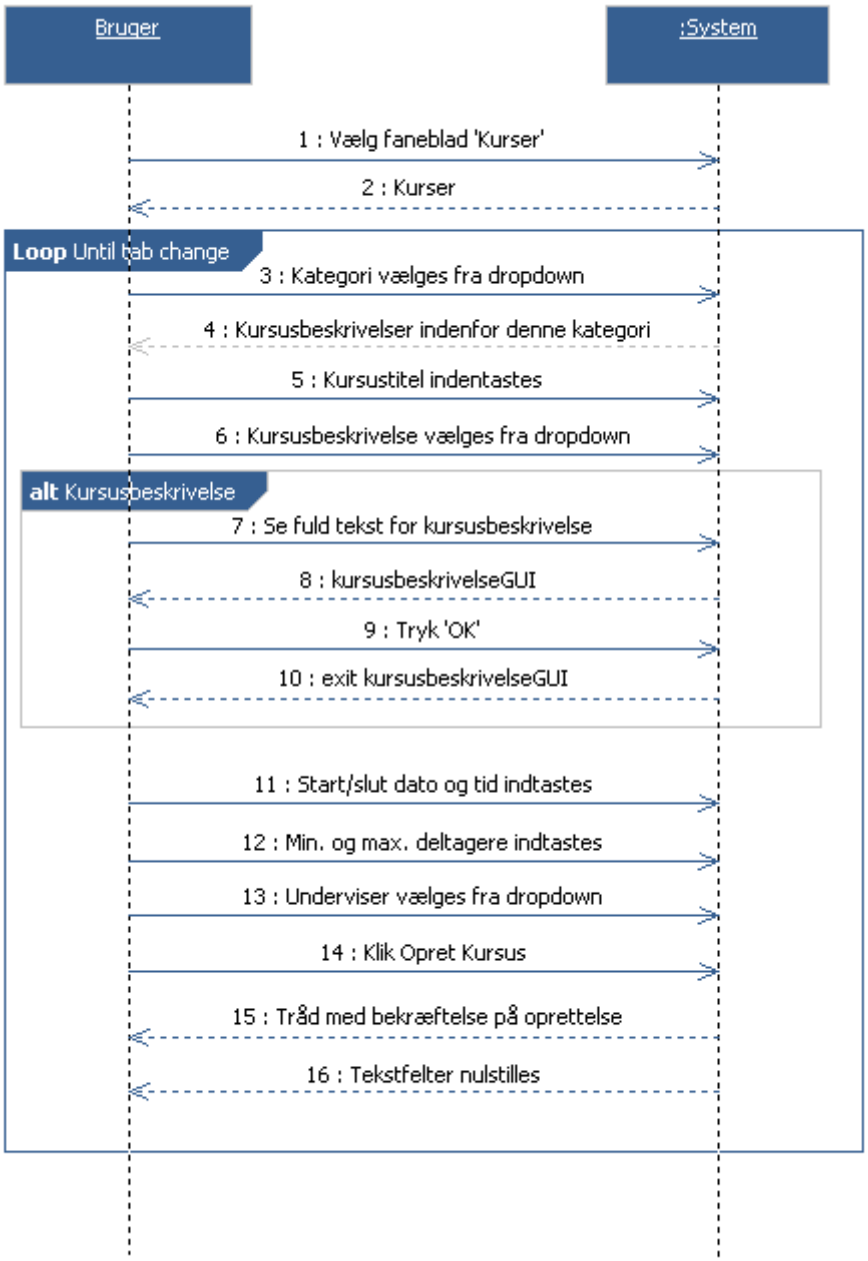
UC. 30
Opret faktura
En faktura oprettes med fakturanr, deltagernavn, dato, beløb, kundenavn, CVR-nr.



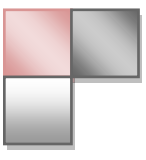
System sekvens diagram

af Daniel og Martin

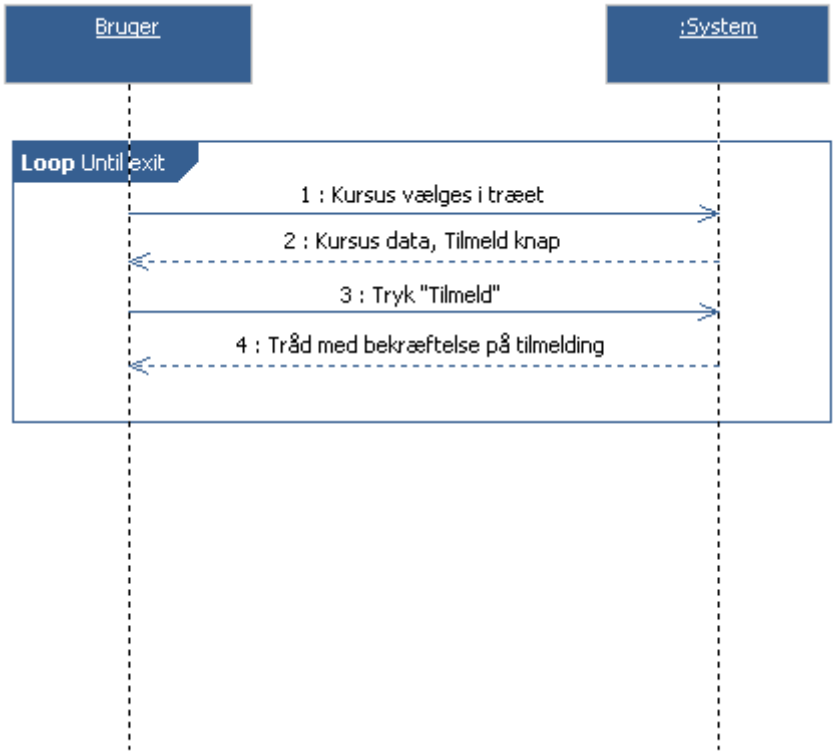
UC1: Opret Kursus



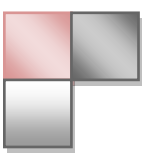
Figur 8.



UC24: Deltagertilmelding til kursus



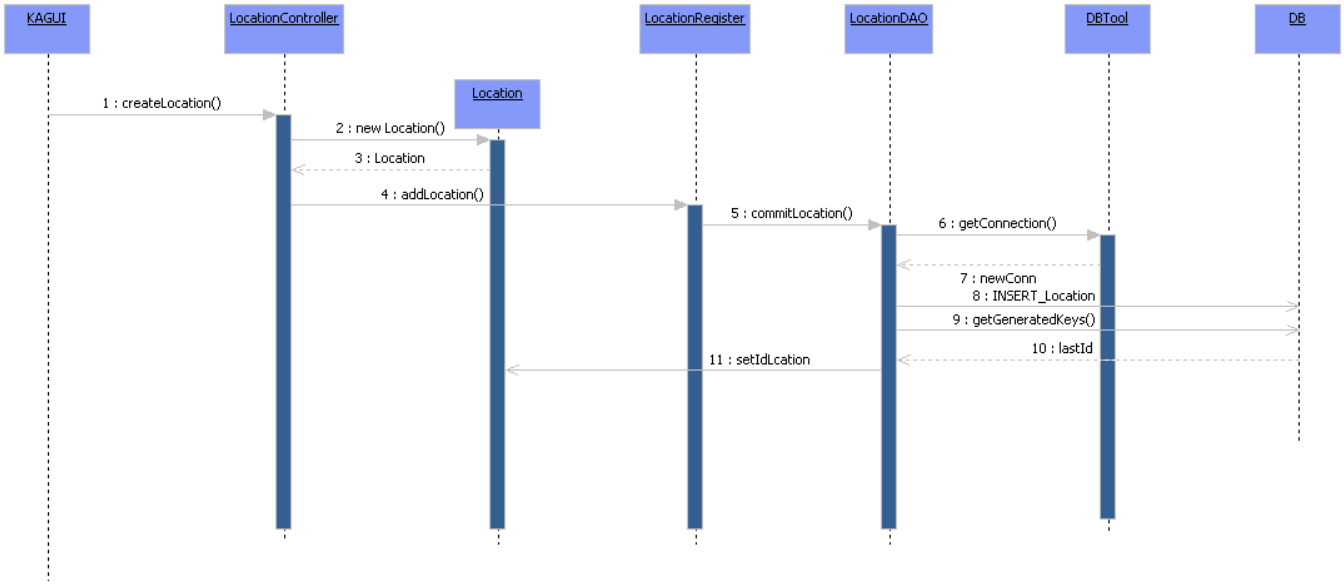
Figur 9.



Sekvens diagram

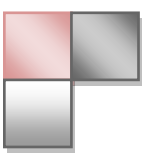
Af Stine og Marianne

Use case 13: Opret kursuslokation

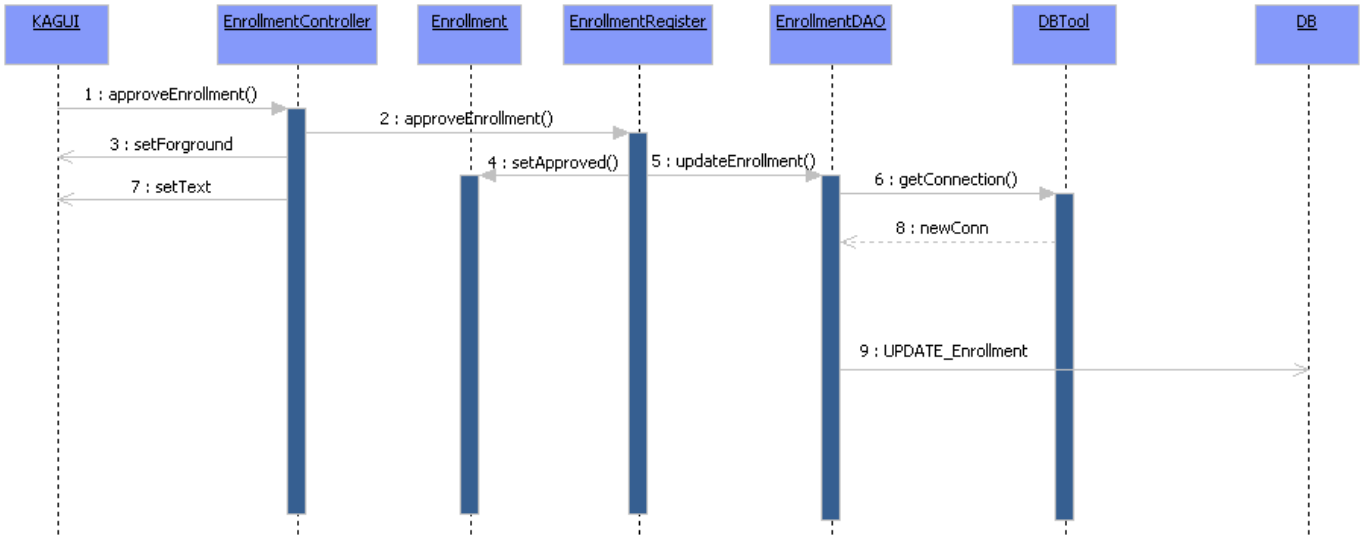


Figur 10.

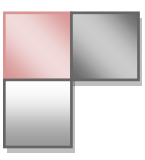
Metodekaldet går igennem alle lag til databasen. Metoderne har parametre, men vi har dem ikke med i dette diagram, da det program(StarUML) vi har brugt ikke tager dem med.



Use case 28: Godkend tilmeldning



Figur 11.



Supplementary Specification

af Daniel

FURPS+

FURPS-modellen identificerer vores systemkrav i kategorier.

Functionality

Administrering af kurser og tilmelding dertil, vist i brugergrænseflader.

Logging and Error Handling

Fejl skal håndteres og behandles af systemet. GUI'en skal vise en fejlmeddelelse ved fejlhåndtering.

Security

Vi vil ikke lægge vægt på sikkerheden da det er et enkeltbrugersystem og skal køres på en lokal computer. Et login kunne overvejes til brugerautentifikation men det har vi valgt at afvige fra da BEC med stor sandsynlighed allerede har et fungerende loginssystem, og derfor er der ingen grund til at udvikle et nyt.

Usability

I deltagertilmeldings-delen af systemet skal der lægges stor vægt på brugervenlighed og overskuelighed. Antallet af trin for at tilmelde sig et kursus skal være så få som muligt.

Reliability

Systemet skal være pålideligt og stabilt. Dermed menes at i tilfælde af system- eller forbindelsesnedbrud, skal ændringer af data ikke gå tabt.

Performance

Funktionerne i systemet skal foregå flydende. Derfor skal svartider til databasen optimeres.

Supportability

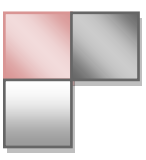
Vi vil udvikle systemet efter Larmans standarder for objektorienteret systemudvikling for at skabe høj binding og lav kobling mellem systemets klasser og dermed sikre systemets fremtidige udvikling.

Implementation Constraints

Systemet skal udvikles i Java.

Free Open Source Components

Java er open source, så vi slipper for produktionsomkostninger som udviklere i forhold til licenser. Derudover har vi valgt at benytte open source databasen MySQL.

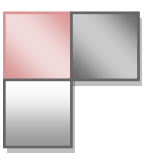


Glossary

af Marianne

Ord	Betydning
Unified Proces (UP)	En objektorienteret systemudviklingsmetode udviklet i slutningen af 1990'erne
System Sekvens Diagram	Interaktion mellem bruger og systemet
Sekvens Diagram	***Viser*** udveksling af meddelelser (dvs. metodekald) mellem flere objekter
GUI eller brugergrænseflade	Den del af systemet som brugeren ser.
BEC	Bankernes EDB-central
SDC	Skandinavisk Data Center
UML	Unified Model Language
SWOT	Strengths, Weaknesses, Opportunities, Threats
HRM	Human Resource Management

Figur 12.



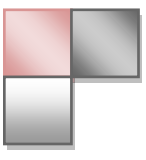
Domain (Business) Rules

af Daniel

Tabellen nedenfor viser en oversigt over forretningsregler, som skal implementeres i systemet i vores BusinessLogic-lag. Kolonnen ID vil henvise til de regler som er implementeret i systemets kode. Kilden definerer om det er en regel sat af virksomheden eller som følge af softwaren.

ID	Regel	Forandringsevne	Kilde
RULE1	Antallet af max. kursister må ikke overskrides for et givent kursus.	Stor. Reglen skal tilpasse sig til hvert kursus.	Applikation
RULE2	Et kursus skal have et minimum af deltagere for at kunne gennemføres.	Stor. Reglen skal tilpasse sig til hvert kursus.	Virksomheden
RULE3	En kursusdeltager må ikke kunne tilmelde sig flere kurser hvor datoerne overlapper.	Lille. Reglen er generel for alle deltagere.	Applikation
RULE4	En underviser må ikke tildeles flere kurser hvor datoerne overlapper.	Lille. Reglen er generel for alle undervisere.	Applikation
RULE5	Uddaterede kurser skal ikke vises i brugergrænsefladen for kursusdeltageren.	Lille.	Applikation
RULE 6	Kurser uden ledige pladser, vises ikke i kursusdeltagerens GUI.	Lille.	Applikation

Figur 13.



Domain model

af Martin

Domænemodellen er en model man benytter i Elaboration fasen for at få overblik over hvordan systemet kommer til at se ud og hvordan man vil programmere det. Man får et grafisk overblik over de håndgribelige klasser i vores kursusadministrationssystem. De eneste klasser der ikke er med i domænemodellen er software-klasserne. Software-klasserne er vores kontrolklasser, GUI-klasser og klasser i persistenslaget.

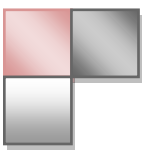
Domænemodellen er bygget af klasser der er koblet sammen med kardinaliteter, altså bindingen mellem klasserne i systemet. En klasse i diagrammet starter med klassenavnet, som altid er skrevet med stort. Hvis klassen er skrevet med kursiv betyder dette at klassen er abstrakt, som er tilfældet med Person klassen. Derefter er klassens attributter nævnt samt tilgængeligheden på de nævnte altså om de er private, public osv. Dette er markeret med et plus for public og et minus for private. Forskellen fra domænemodellen og vores designklassediagram er, at der ikke er skrevet metoder på klasserne.

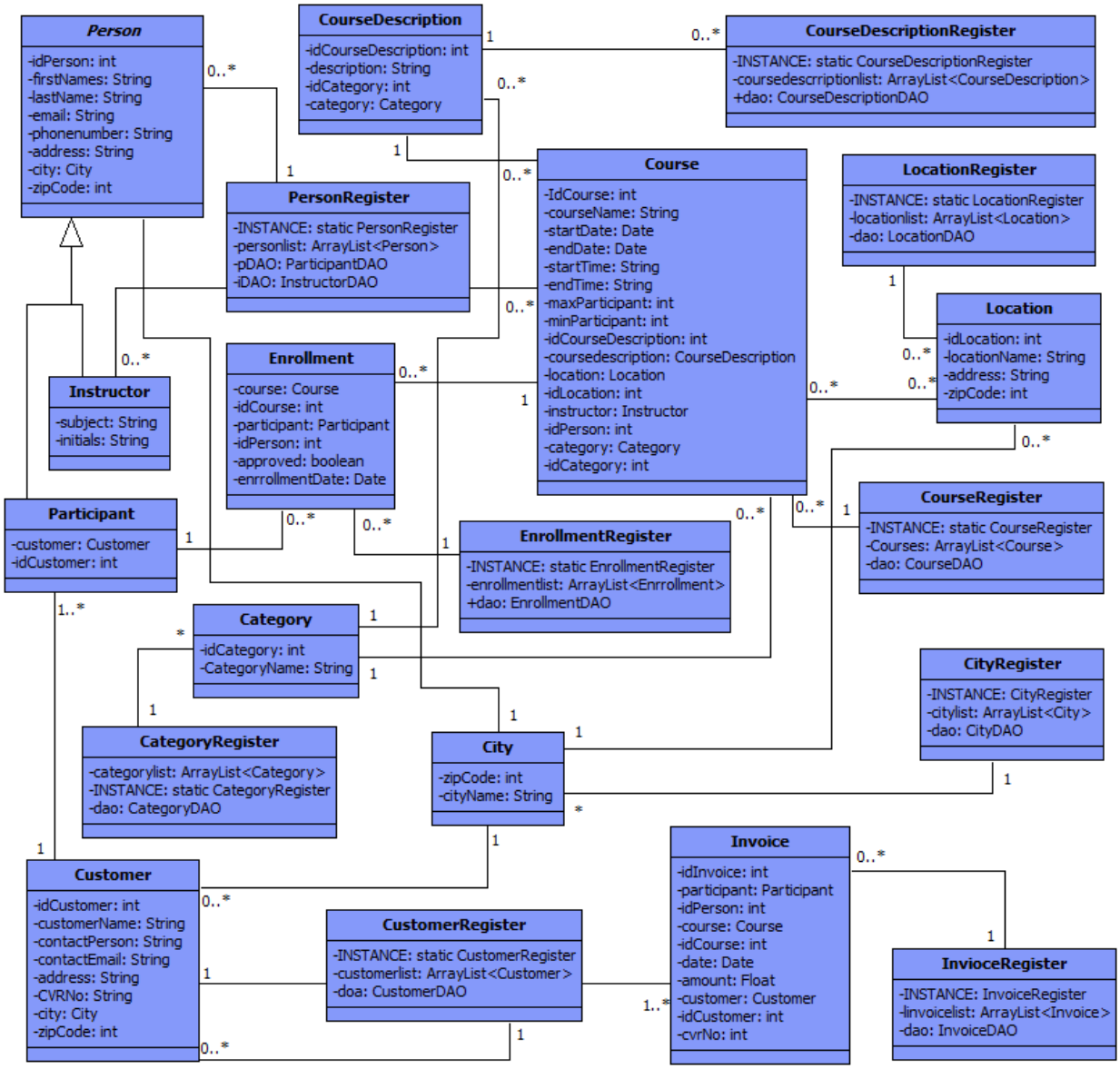
Vores model er designet ud fra vores de use cases vi skrev om i Inception-fasen. Vi har sat kardinaliteter på klassernes kobling. Det symboliserer de små tal ud fra klasserne. Hvis attributten er private, er dette symboliseret med et minus og et plus for public. Vi har tilføjet kardinaliteter på klasserne. Dette er symboliseret med små tal ved siden af en klasse. Tabellen nedenunder forklarer betydningen af kardinaliteterne.

Kardinalitet	Betydning
0..*	0 til mange
1	1
1..*	1 til mange
*	Mange

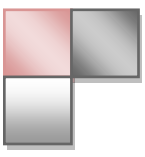
Figur 14.

Et eksempel kunne være forbindelsen mellem CourseDescription og Course. Et kursus kan kun have én kursusbeskrivelse, men en kursusbeskrivelse kan godt have mange kurser. Vi har designet vores system til at have en abstrakt klasse. Den klasse er klassen Person. Der kan ikke oprettes instanser af abstrakte klasser, men de to subklasser: Participant og Instructor, arver attributter fra Person-klassen og herfra opretter der instanser. Dette er vist med en arvepil fra subklasserne til Person.




Person
-idPerson: int
-firstNames: String
-lastName: String
-email: String
-phonenumber: String
-address: String
-city: City
-zipCode: int
CourseDescription
-idCourseDescription: int
-description: String
-idCategory: int
-category: Category
CourseDescriptionRegister
-INSTANCE: static CourseDescriptionRegister
-coursedescriptionlist: ArrayList<CourseDescription>
+dao: CourseDescriptionDAO
PersonRegister
-INSTANCE: static PersonRegister
-personlist: ArrayList<Person>
-pDAO: ParticipantDAO
-iDAO: InstructorDAO
Course
-IdCourse: int
-courseName: String
-startDate: Date
-endDate: Date
-startTime: String
-endTime: String
-maxParticipant: int
-minParticipant: int
-idCourseDescription: int
-coursedescription: CourseDescription
-location: Location
-idLocation: int
-instructor: Instructor
-idPerson: int
-category: Category
-idCategory: int
LocationRegister
-INSTANCE: static LocationRegister
-locationlist: ArrayList<Location>
-dao: LocationDAO
Location
-idLocation: int
-locationName: String
-address: String
-zipCode: int
Instructor
-subject: String
-initials: String
Enrollment
-course: Course
-idCourse: int
-participant: Participant
-idPerson: int
-approved: boolean
-enrrollmentDate: Date
Participant
-customer: Customer
-idCustomer: int
EnrollmentRegister
-INSTANCE: static EnrollmentRegister
-enrollmentlist: ArrayList<Enrrollment>
+dao: EnrollmentDAO
CourseRegister
-INSTANCE: static CourseRegister
-Courses: ArrayList<Course>
-dao: CourseDAO
Category
-idCategory: int
-CategoryName: String
CityRegister
-INSTANCE: CityRegister
-citylist: ArrayList<City>
-dao: CityDAO
CategoryRegister
-categorylist: ArrayList<Category>
-INSTANCE: static CategoryRegister
-dao: CategoryDAO
City
-zipCode: int
-cityName: String
Invoice
-idInvoice: int
-participant: Participant
-idPerson: int
-course: Course
-idCourse: int
-date: Date
-amount: Float
-customer: Customer
-idCustomer: int
-cvrNo: int
Customer
-idCustomer: int
-customerName: String
-contactPerson: String
-contactEmail: String
-address: String
-CVRNo: String
-city: City
-zipCode: int
CustomerRegister
-INSTANCE: static CustomerRegister
-customerlist: ArrayList<Customer>
-doa: CustomerDAO
InvioceRegister
-INSTANCE: InvoiceRegister
-invoicelist: ArrayList<Invoice>
-dao: InvoiceDAO

Figur 15

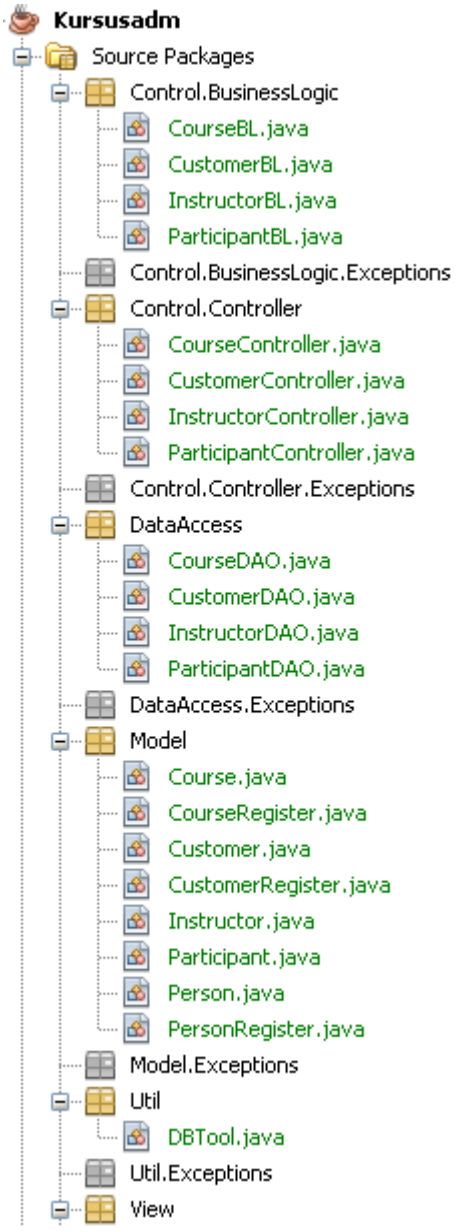


Prototyper

af Daniel

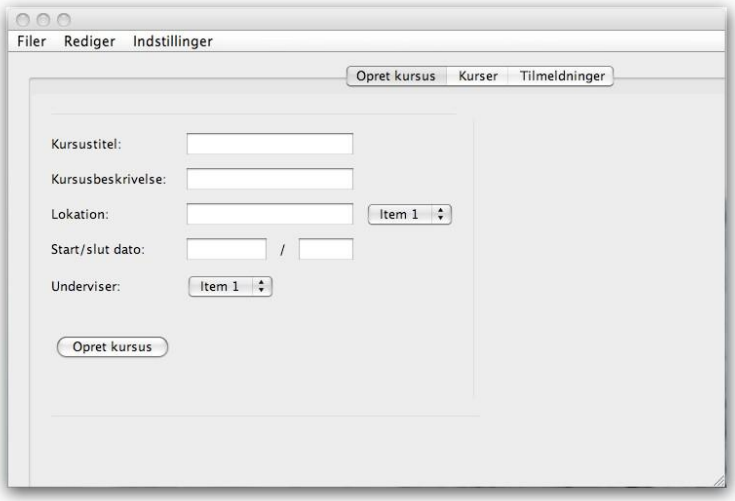
Arkitektur prototype

Arkitekturprototypen er et screenshot af vores MVC prototype artikektur fra NetBeans, som vil blive nærmere beskrevet i afsnittet om Systemarkitektur.



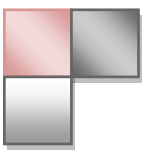
Figur 16.

GUI Prototype



Figur 17.

GUI'en er en prototype af den kursusansvarliges skærmbillede, der vil blive navigeret til forskellige skærmbilleder ved hjælp af faneblade. Dropdown-boksene bliver anvendt til at hente indhentede data.



Systemarkitektur

af Daniel

Anvendelse af trelagsarkitektur (MVC)

Trelagsarkitekturen, også kendt som MVC-modellen (Model, View, Control) er et designmønster der overordnet adskiller klassernes ansvar i tre dele.

Model – Har ansvaret for at modellere objekter fra *virkeligheden* (læs: Domænemodel). Derudover indeholder den data om objekterne gemt i registre.
View – Giver en visuel præsentation af modelklasserne.
Control – Håndterer bruger-input og manipulerer model baseret på input.

Vi har ved hjælp af domænemodellen identificeret alle de klasser Model-laget skulle indeholde.
Vi har for hver af domæneklasserne (udover sub-klasserne) lavet registre hvori objekter hentes ind eller gemmes før de kommer videre til databasen.

I Control-laget har vi lavet en Control-klasse for hver domæneklasse. Laget skal håndtere manipulering af objekter igennem View-laget. Som noget ekstra har vi lavet et Business Logic-lag der skal indeholde alle vores forretningsregler for domæneområderne (Se afsnit om Domain (business) rules). Det vil i praksis bruges ved manipulation af et objekt. Før en manipulationen kan gennemføres, testes der om det opfylder forretningsbetingelsen, er det ikke tilfældet udføres manipulationen (metoden) ikke.

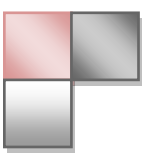
I View-laget er der en klasse for hvert vindue i brugergrænsefladen.

GRASP

Af Martin

GRASP designprincipper er en del af de design mønstre, som vi har udviklet vores kursusadministrationssystem efter. GRASP er en forkortelse for General Responsibility Assignment Software Patterns eller Principles. Anvendelsen af GRASP er en fordel, da der er flere principper i GRASP, end dem beskrevet nedenfor, men vi er gået i dybden med de tre. GRASP medvirker til at der er ansvarsdeling mellem objekterne og der ligger en overvejelse ved hver en klasse i systemet. De følgende elementer i GRASP der har været principper i vores løbende udvikling er:

- Information Expert
- Low coupling
- High Cohesion



Lav kobling klasserne imellem

Vi har udviklet vores system løbende, med tanken i baghovedet at der skal være lav kobling mellem klasserne, dvs. at vi har designet vores klasser til at være mindst muligt afhængige af hinanden. Fordelen ved dette er at hvis vi skal foretage en ændring i en klasse, reducerer vi antallet af påvirkede klasser af denne ændring.

I vores system har vi valgt at uddelegere ansvaret for at f.eks. skabe og initialisere et objekt, da det både er vores register-klasse og control-klassen der ligeligt deler det ansvar. Ulempen ligger idet man laver en forandring af klasserne, ved at konstruktøren initialiserer en skabelse af et objekt af klassen, gennem alle tilhørende klasser op i controlleren. Klassen initialiserer hver attribut, der er en nødvendighed for en skabelse, hvor en attributændring i create-klassen også vil medføre en attributændring i de tilhørende klasser. Det er en ulempe ved opdatering og vedligeholdelse af systemet, størst hvis forandringen ligger i det nederste lag. Til trods for denne ulempe har vi valgt at designe vores system sådan, da fordelene opvejer ulemperne ved uddelegering.

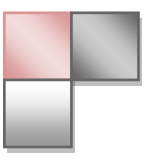
Høj binding

Lav kobling er ikke nok i et godt system. Samtidig med den lave kobling, skal der være høj samhørighed/binding mellem klasserne. Høj samhørighed er klassernes formål og ansvar specialiseret til at omhandle metoder tilegnet til denne gruppe. Hvis en klasse har attributter og metoder der rækker ud over dens ansvar skaber dette dårlig samhørighed, som samtidig vil gå ud over den lave binding. Det vil også give et dårligt overblik over systemdesignet.

For at undgå problemet har vi delt klasserne ind i grupper, der hver styrer sin del af systemet. F.eks. kan vi kigge på vores "model-package" der ligger alle vore model-klasser. Hver af de enkelte er med til at modellere objekter af egen klasse. Objekter i modelklassen initialiserer ikke metoder hos andre grupper.

Information Expert

I vores system er alle vores register-klasser informationseksperter. Betegnelsen anvendes til klasser der indeholder al information for at kunne gennemføre en opgave. Vores register-klasser er informationseksperter, da vi ikke rigtig vil kunne fortage os noget uden at skulle anvende viden som objekterne, der ligger i registret, indeholder for at kunne køre systemet.



Singleton-pattern

Flere steder i systemet har vi valgt at anvende Singleton for at gøre bestemte klasser globalt tilgængelige i programmet og derudover sikre at kun én instans af klassen findes.

Singleton kan anvendes på to måder, eager og lazy.
Ved eager-metoden, opretter man objektet allerede ved systemets opstart (ClassLoaderen) og når man så skal anvende objektet returnerer den objektet med det samme, som vist i kodeeksemplet nedenfor.

```
public class EagerSingleton {
private static EagerSingleton INSTANCE = new EagerSingleton();

private EagerSingleton() {}

public static EagerSingleton getInstance() {
 return INSTANCE;
}
```

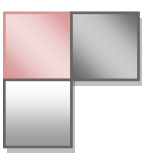
Figur 18.

Ved lazy-metoden oprettes objektet først når singleton-metoden kaldes. Der undersøges om en instans af klassen er oprettet i forvejen ved hjælp af et booleansk udtryk.

```
public class LazySingleton {
private static LazySingleton INSTANCE = null;

private LazySingleton() {}

public static LazySingleton getInstance() {
 if(INSTANCE == null) {
   INSTANCE = new LazySingleton();
 }
 return INSTANCE;
 }
```

Figur 19.

Der er to fælles kendetegn ved de to metoder. Den ene er at konstruktøren altid er private. Dermed forhindrer vi at andre klasser kan oprette en instans af Singleton-klassen.
Den anden er at getInstance metoden er *public static*. Det sikrer at metoden kan kaldes globalt, også kaldet en klassemetode.
Hvilken af metoderne man avender afhænger meget af hvilket system man udvikler. I afsnittet om Utility vil vi vise et eksempel hvor anvendelsen af lazy-metoden er at foretrække.



Persistens-Laget

Persistenslaget er det lag der adskiller al databasehåndtering fra vores MVC arkitektur.
Det vil sige, alle forespørgsler og forbindelser til databasen foregår i dette lag og når aldrig ind til MVC'en.
Formålet med dette er at mindske komplikationer ved eksempelvis at skifte til en ny type af database.

Data Access Object (DAO)

DAO-laget indeholder en DAO-klasse for hver domæneklasse. Alle forespørgsler på databasen udføres og håndteres i dette lag for at undgå database scripts i MVC'en.

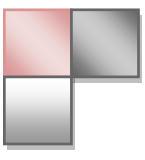
DAO-klassen kan eksempelvis trække data ud fra databasen og omforme det til objekter som derefter kan gemmes i en kollektion (ArrayList).

Et eksempel på en DAO-metode:

```
public ArrayList<CourseDescription> getCourseDescriptionList() {
        ArrayList<CourseDescription> list = new
ArrayList<CourseDescription>();
        try {
            Connection conn = DBTool.getInstance().getConnection();
            PreparedStatement st =
conn.prepareStatement(SELECT_CourseDescriptions);
            ResultSet rs = st.executeQuery();
            while (rs.next()) {
                CourseDescription cd = new CourseDescription();
                cd.setIdCourseDescription(rs.getInt(1));
                cd.setDescription(rs.getString(2));
                cd.setIdCategory(rs.getInt(3));
                list.add(cd);

            }
            conn.close();
        } catch (SQLException ex) {

Logger.getLogger(CourseDescriptionDAO.class.getName()).log(Level.SEVERE,
null, ex);
        }
        return list;
    }
```

Figur 20.

Denne metode returnerer en kollektion af alle kursusbeskrivelser i databasen. Der bliver oprettet forbindelse til databasen og der eksekveres en forespørgsel dertil. Den løber en løkke igennem der skaber et nyt kursusbeskrivelse-objekt for hver række i tabellen der er hentet ind i result-sættet. For hvert objekt bliver der kaldt set-metoder som henter data fra kolonneindeks i result-sættet. Når løkken er færdig lukkes forbindelsen til databasen og der returneres en liste af kursusbeskrivelser.

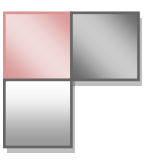


Utility

Utility indeholder klassen DBTool, det er en forholdsvis enkel klasse som blot skal håndtere forbindelsen til databasen. Klassen henter adressen samt logininformationer til databasen i en Property-fil. Det gør at der let kan ændres adresse (til en anden MySQL database) eller login uden at skulle rekompilere koden.

Nedenfor er metoden vist som returnerer et Connection-objekt.

```
public Connection getConnection() throws SQLException {

        Properties props = new Properties();

        try {

            Class.forName("com.mysql.jdbc.Driver");

            props.load(new FileInputStream("db.properties"));

        } catch (IOException ex) {

            Logger.getLogger(DBTool.class.getName()).log(Level.SEVERE,
null, ex);

        } catch (ClassNotFoundException ex) {

            Logger.getLogger(DBTool.class.getName()).log(Level.SEVERE,
null, ex);

        }

Connection newConn =
DriverManager.getConnection(props.getProperty("url"),
props.getProperty("username"), props.getProperty("password"));

        if (!newConn.isClosed()) {

            System.out.println("Successfully connected to MySQL
Database...");

        }

        return newConn;

    }
```

Eager / Lazy-loading af objekter fra databasen

af Daniel og Martin

Der er tre forskellige måder at indlæse objekter fra databasen på.

- Eager/Eager: Alle objekter læses ind ved applikationens opstart og associeringerne skabes.
- Eager/Lazy: Alle objekterne indlæses ved applikationens opstart, men associeringerne skabes først for det givne objekt når det skal anvendes.
- Lazy/Lazy: Objekterne og dets associationer indlæses først når det skal anvendes (bliver bl.a. anvendt som default i Hibernate.)

Lazy-loading bidrager til performance ved at minimere optagning af ressourcer.

Eager-loading bidrager til performance ved at minimere kommunikationen mellem system og database.

Vi har i denne version af systemet valgt at anvende eager-loading af objekterne fra databasen. Argumentet for dette valg var at satse på en *måske* langsommere opstartstid af applikationen, men derefter en hurtig eksekvering af systemets funktioner.

Vi skaber referencer mellem objekterne vha. indlæsning af fremmednøglerne. Ud fra nøglerne kan vi derefter sætte associationerne.

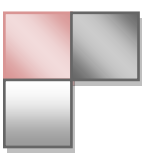
Som vist i Figur 20 indlæses alle kursusbeskrivelse-objekter med fremmednøglen "idcategory", som er primærnøgle i tabellen Category, men associeringen til Category-objektet bliver ikke skabt endnu. Nedenfor er metoden hvor vi skaber associeringerne mellem kategori- og kursusbeskrivelsesobjekter.

```
  public void createCourseDescriptionAssociations() {
        for (CourseDescription cd : coursedescriptionlist) {

            int idCategory = cd.getIdCategory();
            Category c =
CategoryRegister.getInstance().getCategoryFromId(idCategory);
            cd.setCategory(c);
        }
    }
```

Figur 22.

Metoden ligger i kursusbeskrivelse-register klassen da den som før nævnt er Information Expert.
For hvert kursusbeskrivelse-objekt gemmer vi fremmednøglen idcategory i en lokal variabel.
Derefter bruges den lokale variabel som argument i getCategoryFromId-metode, som løber en løkke igennem kategori-registret og returnerer det objekt der matcher med denne nøgle.
Sidste udførelse i løkken er at associeringen sættes med det returnerede kategori-objekt.

Da vi som vist ovenfor har benyttet eager-eager designet, skal vi ikke tage højde for i hvilken rækkefølge associeringerne dannes.



Vi anvender en init metode til at eksekvere vores eager-eager strategi. Denne initialiseringsmetode bliver kaldt af konstruktøren i vores GUI.

Et uddrag af initaliseringsmetoden – initializeData():

```
   CategoryRegister.getInstance().fillCategories();
        CourseDescriptionRegister.getInstance().fillCourseDescriptions();

        ……..

CourseDescriptionRegister.getInstance().createCourseDescriptionAssociatio
ns();
```

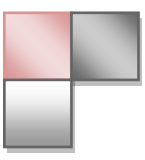
Figur 23.

Et kodesegment af metoden fillCourseDescriptions() fra klassen CourseDescriptionRegister:

```
        public void fillCourseDescriptions() {
        coursedescriptionlist.addAll(dao.getCourseDescriptionList());
         }
```

Figur 24.

I metoden fillCourseDescriptions() tilføjer vi en kollektion til vores ArrayList af kursusbeskrivelsesobjekter. Vi bruger metoden getCourseDescriptionList(), som ligger i klassen CourseDescriptionDAO (Figur 20), som argument.



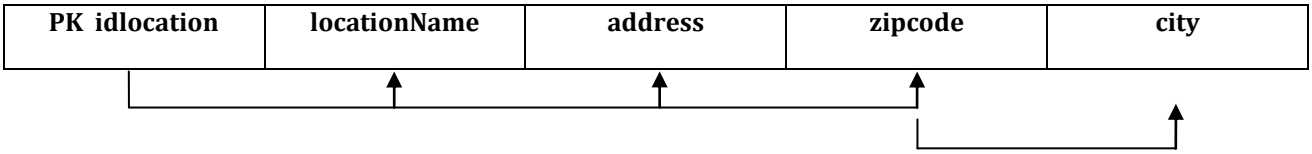
Normalisering

af Marianne

Kilde: Database Solutions, Thomas Conolly & Carolyn Begg, Pearson.

Normalisering af databaser bruges til at sikre at der ikke er redundans i tabellerne. Ligeledes bruges det til at optimere gennemløb og lette vedligehold af databasen. Inden for databasedesign findes flere normalformer men de mest anvendte hedder 1., 2. og 3. normalform(NF). Målet ved normalisering er at bringe alle tabeller op på 3. normalform.

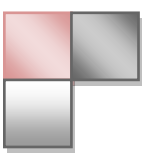
Et eksempel:



Denne tabel er allerede på 2. normalform, da 1. normalform betyder at alle værdier er atomiske, dvs. de kan ikke deles og der er kun én værdi i hver tupel. I dette tilfælde er idlocation primærnøgle og udpeger locationName, address og zipcode. Tabellen er ikke på 3. normalform fordi der er transitiv afhængighed mellem zipcode og city, dvs. zipcode udpeger city. For at bringe tabellen op på 3. normalform splittes den op i 2 tabeller og zipcode bliver så til fremmednøgle(FK) i den første tabel og primærnøgle(PK) i den nye tabel. Det ser således ud:

PK idlocation	locationName	Address	FK zipcode

PK zipcode	city



Design Class Diagram

Af Stine

Et Design Class Diagram(DCD) ligner en Domæne Model(DM), og er som sådan også en udbygning af denne. Alle de klasser der er i en DM vil også være i et DCD.
En klasse er delt op i tre afdelinger; øverst klassenavnet, som, hvis det står i kursiv, er abstrakt. Feltet i midten indeholder atributterne, som i de fleste tilfælde(i vores projekt alle) er private. I sidste felt er metoderne, der i de fleste tilfælde er public.
Diagrammet viser hvordan alle klasser incl. softwareklasser, interagerer med hinanden.

Vi bruger flere typer associationslinier i vores DCD; Den mest brugte er en ubrudt linie mellem to klasser, en sådan linie beskriver et forhold mellem to klasser hvor den ene klasse eksempelvis bruger metoder som den anden har.

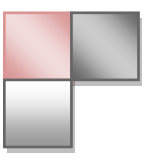
En linie med en fyldt diamant i den ene ende er en "composite aggreagation" som er en stærk type af association. Klassen i diamantenenden, er ansvarlig for skabelse og sletning af instancer af klassen i den anden ende. Eks. CourseRegister-klassen har metoderne til at slette og oprette objekter af Course-klassen.

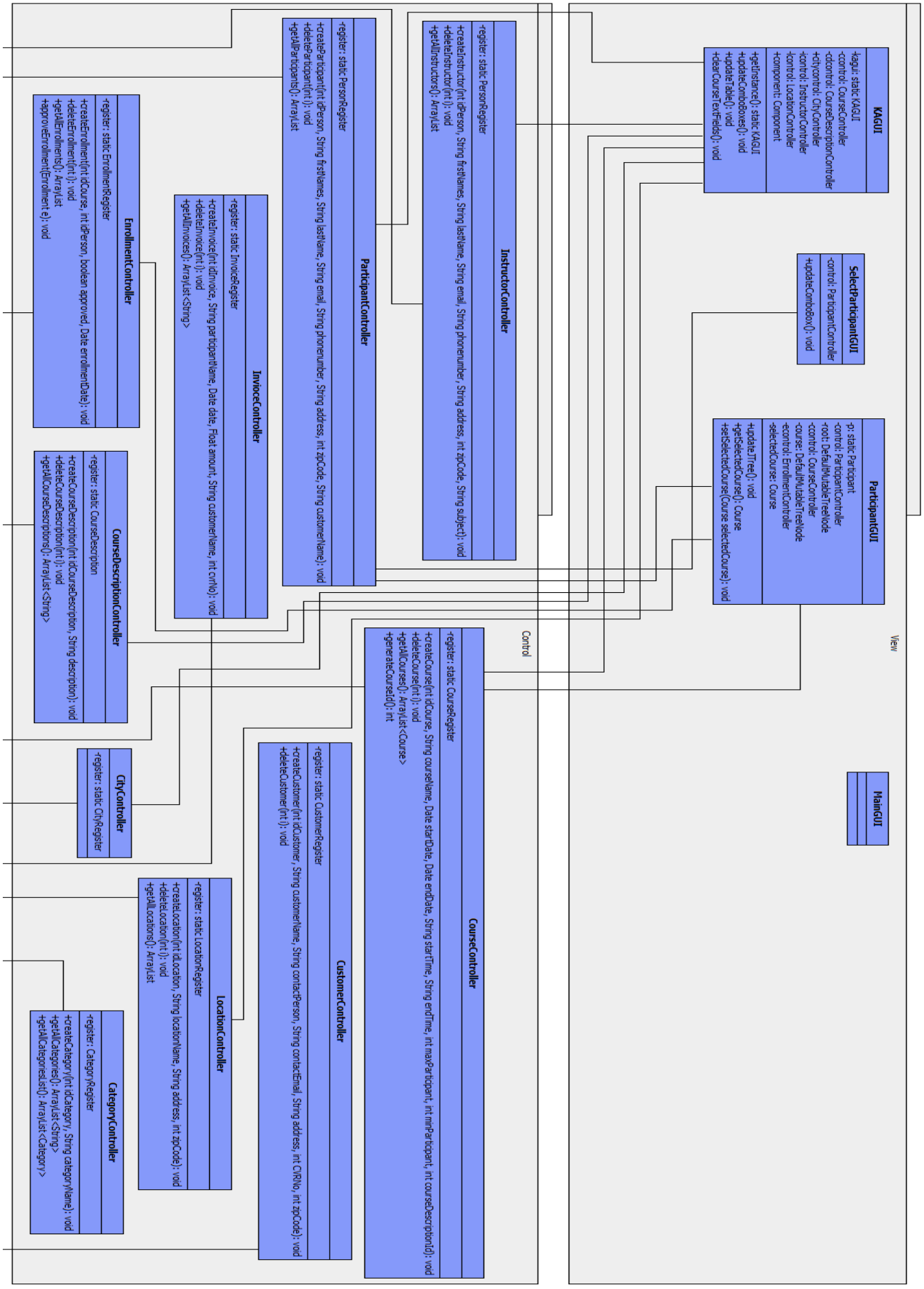
Klasserne Instructor og Participant er subklasser der er specialiseret af klassen Person(superklasse), de nedarver begge Person-klassens atributter men har selv nogle individuelle tilføjet. Hvis generaliseringsklassen(Person)slettes, bliver de to specialiseringsklasser(Participant, Instructor) også slettet.
PersonRegister-klassens ArrayList er af typen Person, men objekterne er typecastede til Instructor og Participant.

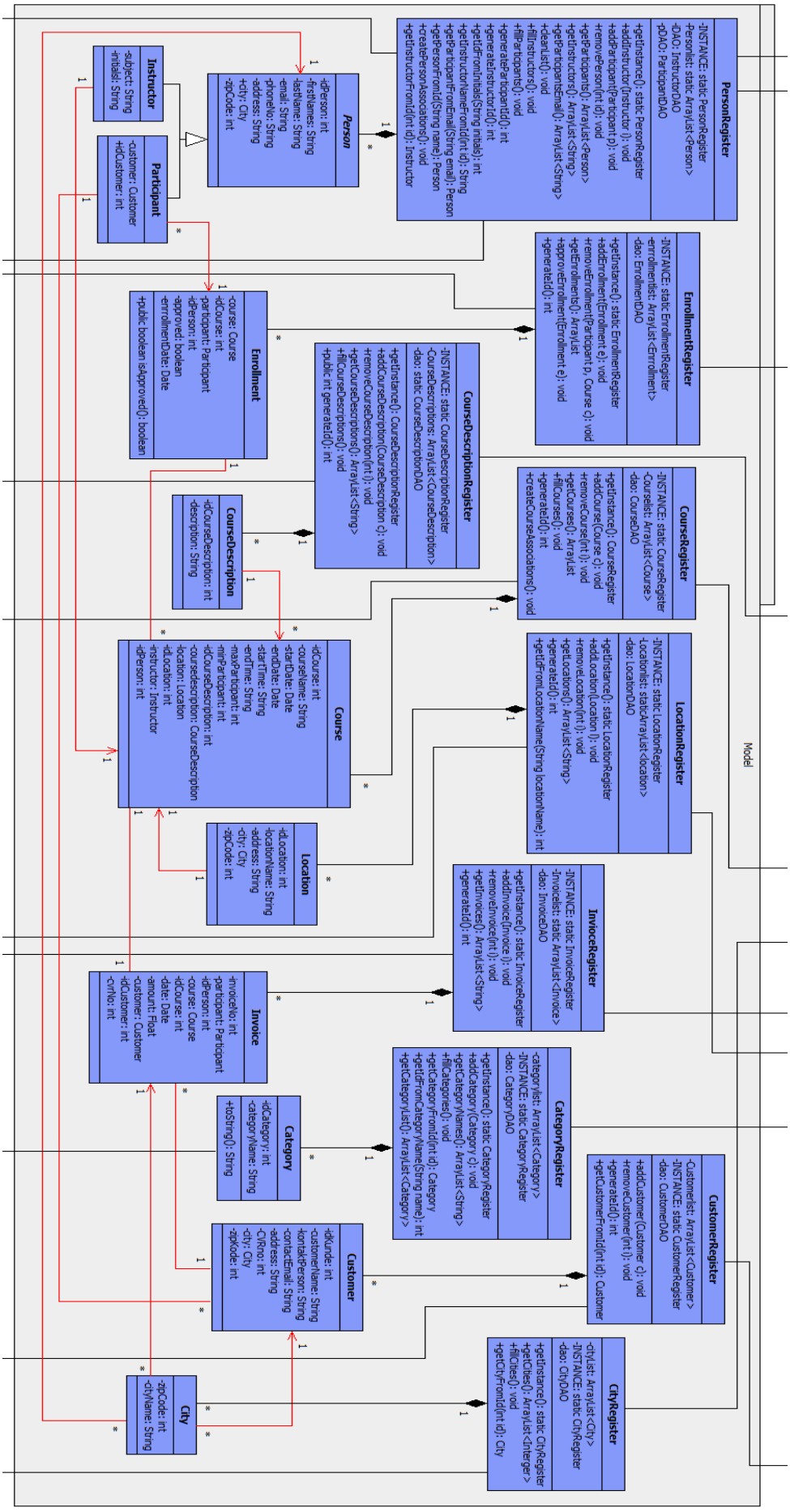
Den sidste version af associationslinier vi har brugt, er en ubrudt linie med en lille pil i den ene ende, den viser blot hvilken vej associationen går, altså en retningsviser.

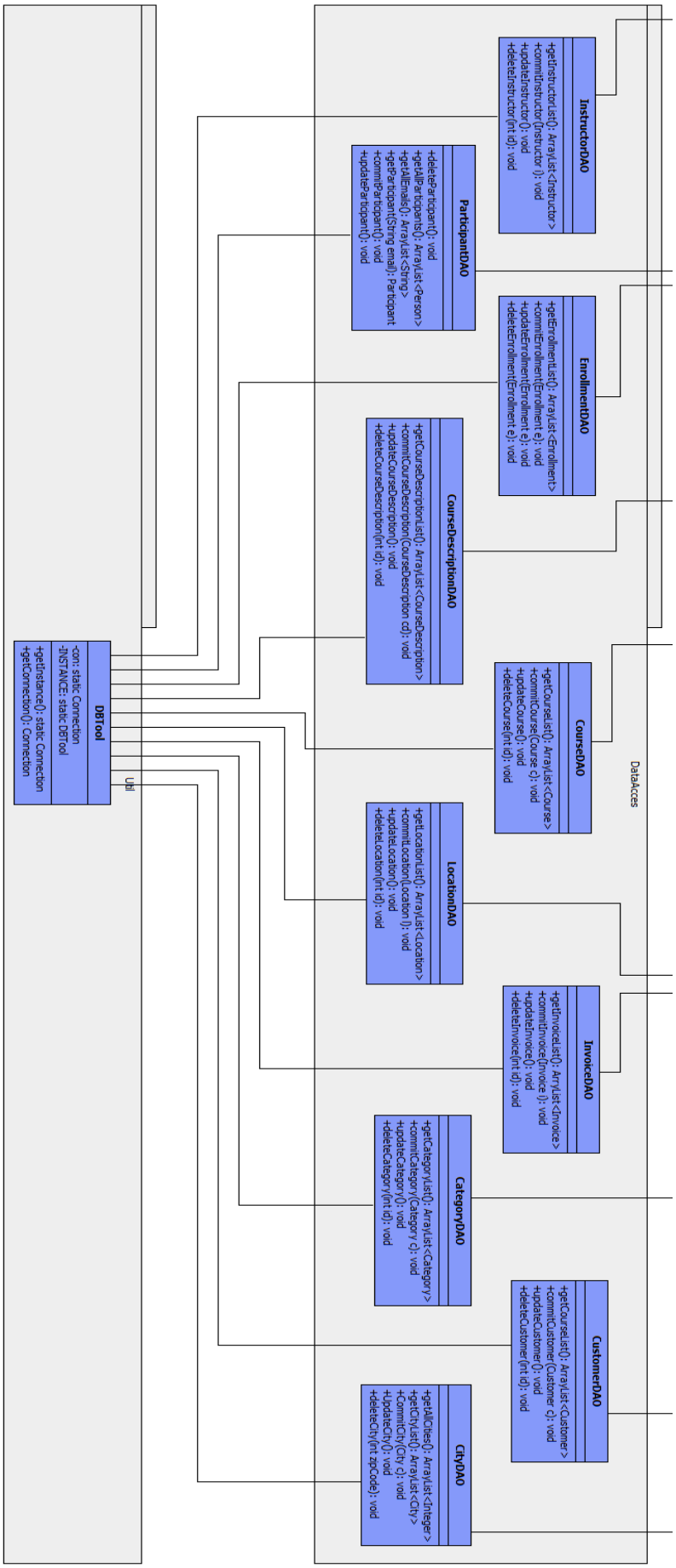
Multiplicity, eller Kardinalitet, viser antallet af objekter af denne klasse der interagerer med den anden klasses objekt/objekter.
Eks. Forholdet mellem Course, Enrollment og Participant, en deltager(Participant) kan have flere tilmeldinger (til flere kurser), men kun en tilmelding pr. kursus pr. deltager.

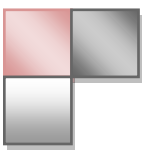
NB.: En A3 version af Design Class Diagrammet er vedlagt som bilag.




KAGUI
-kagui: static KAGUI
-ccontrol: CourseController
-cdcontrol: CourseDescriptionController
-ctycontrol: CityController
-icontrol: InstructorController
-lcontrol: LocationController
-component: Component
+getInstance(): static KAGUI
+updateComboBoxes(): void
+updateTable(): void
+clearCourseTextFields(): void
SelectParticipantGUI
-control: ParticipantController
+updateComboBox(): void
ParticipantGUI
-p: static Participant
-control: ParticipantController
-root: DefaultMutableTreeNode
-ccontrol: CourseController
-course: DefaultMutableTreeNode
-econtrol: EnrollmentController
-selectedCourse: Course
+updateTree(): void
+getSelectedCourse(): Course
+setSelectedCourse(Course selectedCourse): void
InstructorController
-register: static PersonRegister
+createInstructor(int idPerson, String firstNames, String lastName, String email, String phonenumber, String address, int zipCode, String subject): void
+deleteInstructor(int): void
+getAllInstructors(): ArrayList
ParticipantController
-register: static PersonRegister
+createParticipant(int idPerson, String firstNames, String lastName, String email, String phonenumber, String address, int zipCode, String customerName): void
+deleteParticipant(int): void
+getAllParticipants(): ArrayList
EnrollmentController
-register: static EnrollmentRegister
+createEnrollment(int idCourse, int idPerson, boolean approved, Date enrollmentDate): void
+deleteEnrollment(int): void
+getAllEnrollments(): ArrayList
+approveEnrollment(Enrollment e): void
InvioceController
-register: static InvioceRegister
+createInvoice(int idInvoice, String participantName, Date date, Float amount, String customerName, int cvrNo): void
+deleteInvoice(int): void
+getAllInvoices(): ArrayList<String>
CourseDescriptionController
-register: static CourseDescription
+createCourseDescription(int idCourseDescription, String description): void
+deleteCourseDescription(int): void
+getAllCourseDescriptions(): ArrayList<String>
CityController
-register: static CityRegister
CourseController
-register: static CourseRegister
+createCourse(int idCourse, String courseName, Date startDate, Date endDate, String startTime, String endTime, int maxParticipant, int minParticipant, int courseDescriptionId): void
+deleteCourse(int): void
+getAllCourses(): ArrayList<Course>
+generateCourseId(): int
CustomerController
-register: static CustomerRegister
+createCustomer(int idCustomer, String customerName, String contactPerson, String contactEmail, String address, int CVRNo, int zipCode): void
+deleteCustomer(int): void
LocationController
-register: static LocationRegister
+createLocation(int idLocation, String locationName, String address, int zipCode): void
+deleteLocation(int): void
+getAllLocations(): ArrayList
CategoryController
-register: CategoryRegister
+createCategory(int idCategory, String categoryName): void
+getAllCategories(): ArrayList<String>
+getAllCategoriesList(): ArrayList<Category>
MainGUI
View
Control


Model
PersonRegister
Person
Instructor
Participant
EnrollmentRegister
Enrollment
CourseDescriptionRegister
CourseDescription
CourseRegister
Course
LocationRegister
Location
InvoiceRegister
Invoice
CategoryRegister
Category
CustomerRegister
Customer
CityRegister
City


Util
DBTool
-con: static Connection
-INSTANCE: static DBTool
+getInstance(): static Connection
+getConnection(): Connection
DataAcces
InstructorDAO
+getInstructorList(): ArrayList<Instructor>
+commitInstructor(Instructor i): void
+updateInstructor(): void
+deleteInstructor(int id): void
ParticipantDAO
+deleteParticipant(): void
+getAllParticipants(): ArrayList<Person>
+getAllEmails(): ArrayList<String>
+getParticipant(String email): Participant
+commitParticipant(): void
+updateParticipant(): void
EnrollmentDAO
+getEnrollmentList(): ArrayList<Enrollment>
+commitEnrollment(Enrollment e): void
+updateEnrollment(Enrollment e): void
+deleteEnrollment(Enrollment e): void
CourseDescriptionDAO
+getCourseDescriptionList(): ArrayList<CourseDescription>
+commitCourseDescription(CourseDescription cd): void
+updateCourseDescription(): void
+deleteCourseDescription(int id): void
CourseDAO
+getCourseList(): ArrayList<Course>
+commitCourse(Course c): void
+updateCourse(): void
+deleteCourse(int id): void
LocationDAO
+getLocationList(): ArrayList<Location>
+commitLocation(Location l): void
+updateLocation(): void
+deleteLocation(int id): void
InvoiceDAO
+getInvoiceList(): ArrayList<Invoice>
+commitInvoice(Invoice i): void
+updateInvoice(): void
+deleteInvoice(int id): void
CategoryDAO
+getCategoryList(): ArrayList<Category>
+commitCategory(Category c): void
+updateCategory(): void
+deleteCategory(int id): void
CustomerDAO
+getCourseList(): ArrayList<Customer>
+commitCustomer(Customer c): void
+updateCustomer(): void
+deleteCustomer(int id): void
CityDAO
+getAllCities(): ArrayList<Integer>
+getCityList(): ArrayList<City>
+CommitCity(City c): void
+UpdateCity(): void
+deleteCity(int zipCode): void



Testplan

Af Daniel og Martin

Der findes flere forskellige former af tests på forskellige niveauer.

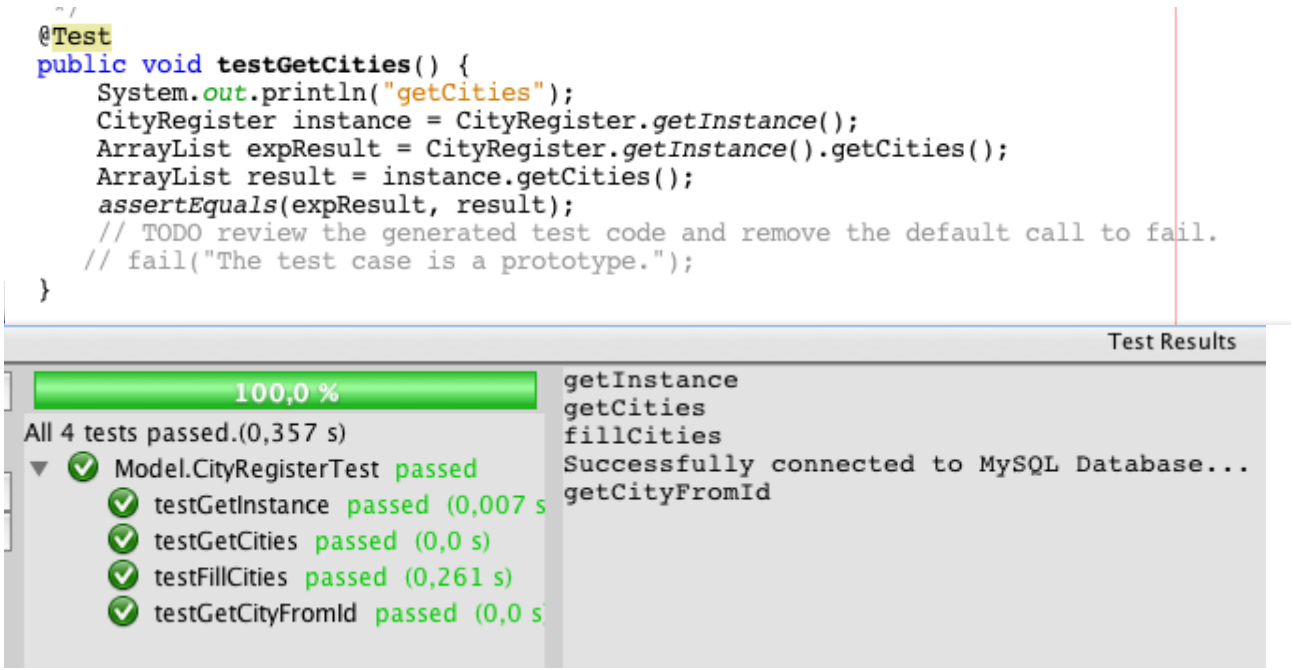
- Whitebox-Test
- Blackbox-Test
- JUnit-Test
- Usability-Test

Vi vil igennem udviklingen af systemet løbende bruge Whitebox-testen. Vi benytter det for at sikre os at funktionaliteten i systemet fungerer.

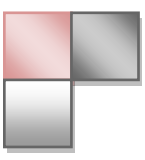
JUnit

Med JUnit-test kan man opstille en test case af alle ens metoder. Man kan teste for fejl og få et detaljeret overblik over hvilke metoder der har fejlet og hvilke der er godkendt. Man kan få printet fejlmeddelelser ud på de fejlede metoder. Vi vil opstille en enkelt JUnit-test, for at vise hvordan den kan bruges. Men da vi ikke har tænkt os at bruge JUnit i den løbende udvikling, bliver det ikke videre uddybet. Den fulde JUnit-test af klassen CityRegister er vedlagt som bilag.

Screenshot af en vellykket JUnit-test.



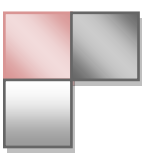
Figur 25.



Brugergrænseflade evaluering

Use Case 1 –	**Opret Kursus**	
Test-user :	Lars Kofod	
Carry out		
Explain purpose		
1. Problem :	Manglende tidsformat	
2. Problem :	I tvivl om kurset var oprettet (tråden med beskeden var ikke tydelig nok)	
Keystrokes and mouseclicks	:	72
Total keystroke time	:	72* 0,2 = 14,4 sekunder
Task-time test	:	3 minutter.
Målt ved 1 test		

Use Case 24 –	**Deltagertilmelding til kursus**	
Test-user :	Lars Kofod	
Carry out		
Explain purpose		
1. Problem :	Testbrugeren ville gerne kunne se det kursus han er tilmeldt til.	
Keystrokes and mouseclicks	:	2
Total keystroke time	:	2* 0,2 = 0,1
Task-time test	:	51 sekunder.
Målt ved 1 test		



Konklusion

af Gruppen

I forhold til vores analyse af BEC, som virksomhed, har vi ikke følt at den har bidraget til en bedre systemudviklingsproces. Dermed mener vi at den ikke har hjulpet med til at identificere krav til et kursusadministrationsssystem. Vi har dog fået et indgående kendskab til virksomheden, som kunne hjælpe os i fremtidig udvikling af andre systemer indenfor denne sektor.
Da vi ikke har haft mulighed for at have en dialog med BEC har virksomhedsanalysen båret præg af dette, i form af at alle informationer er hentet på nettet.

Gennem hele forløbet har vi haft et meget tæt samarbejde i gruppen. Vi har mødtes på skolen alle hverdage samt nogle weekender.
Ideelt var tankegangen at alle arbejdsområder skulle fordeles ligeligt, så vi hver især fik chancen for at udvikle vores kompetencer, det viste sig dog hurtigt at være urealistisk indenfor vores kunnen og tidsramme.
Så længere henne i processen endte vi med at dele opgaverne op inden for de områder hvor vi var bedst egnet.

Igennem de indledende faser har vi lært at vi fremover skal disponere vores tid bedre. Det tog os længere tid end beregnet at komme ordenligt i gang med opgaven, og vi har lært at løsningen til dette problem bliver at vi næste gang skal sætte flere interne deadlines i gruppen.

Der blev brugt meget tid på at diskutere systemarkitektur og løsningsmodeller. Herunder især implementering af en database i en Java-applikation. De største udfordringer vi stødte på i den forbindelse var indlæsning af objekter og associeringerne hertil.

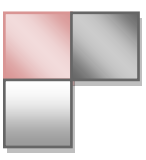
På trods af systemets fejl og mangler mener vi at have fået systemets nøglefunktioner på plads, og at det har givet os en grundlæggende viden indenfor Java og SQL til at kunne implementere de resterende Use Cases, i en Construction-fase.

Næste version

Vi har valgt at nævne hvad vi gerne ser implementeret i næste version af systemet da det som sagt er en prototype og ikke fuldt funktionelt.

I næste version vil der blive implementeret faktuering, evaluering og statistik, som vi på nuværende tidspunkt slet ikke har været omkring i koden. Vores egne exceptions har vi ikke implementeret i denne version, dog har vi sørget for at SQL-exceptions bliver håndteret i DAO-laget, og dermed ikke bliver kastet videre i systemet.

Det kunne også være interessant at tage fat på samtidighedsproblematikken i forbindelse med den næste version.



Litteraturliste

Proffesional Systems Development, Niels Erik Andersen et. Al.

Applying UML and Patterns, C. Larman, Prentice Hall

Database Solutions. Thomas Connolly. Pearson

Div. analysetekniker:

SWOT, PEST, Porter Five, Boston Matrix, ValueChain og ExperienceCurve.

Projekt Etablering. Professionel IT.forundersøgelse, Kjeld Bødger m.fl.

IT and Business Models. J. Hedman & t. Kalling. Liber.

Objects First with Java, A Practical Introduction using BlueJ, David J. Barnes & Michael Kölling

Head First Design Patterns, Eric Freeman, Elisabeth Freeman, Kathy Sierra & Bert Bates