System design af opret ordre for Vestbjerg Byggecenter A/S

Projekt Raport

Projektgruppe 2

Andreas Larsen
Jacob Thomsen
Katrine Vindbæk Madsen
Matias Kastrup Kragh

University College Nordjylland
Datamatiker
Sofiendalsvejs 60
DK-9200 Aalborg SV

University College Nordjylland

Datamatiker
Sofiendalsvej 60
DK-9200 Aalborg SV

Title:
System design af opret ordre for Vestbjerg Byggecenter A/S

Project Period:
December 2017

Project Group:
Projektgruppe 2

Participant(s):
Andreas Larsen
Jacob Thomsen
Katrine Vindbæk Madsen
Matias Kastrup Kragh

Supervisor(s):
Anita Lykke Clemmensen
István Knoll
Torben Larsen

Characters:
43.729 uden mellemrum

Repository path og revision-snr: `https://kraka.ucn.dk/svn/dmab0917_1Sem_Projekt_2` Nr: 88

Page Numbers: 65

Date of Completion:
December 15, 2017

Abstract:

This project report is a documentation of the analysis and development of a system for a fictional company dealing in construction materials such as wood and tools. The developed system is based on an analysis of the company and the information given about the company by a representative. It was tasked to develop, at the least, one use case and document the implementation. The company analysis consisted of a 'return of investment' analysis based on the demands made by the company and a subsequent use case prioritization was conducted. The use case this report largely deals with is in the design phase and implementation of this report.

It was possible to implement the basic functional version of the use case with a functional user interface. An evaluation of the process was also conducted as this was a part of the project as well.

Preface

Læse vejledning

Rapporten kan læses i rækkefølgen afsnittene er opstillede. Kapitler og afsnit er inddelt i 'Parts' for at vise et skift i rapportens overliggende emne.

Kilder er angivet med nummering og kan findes til sidst i rapporten før ordlisten. Bilaget vil indeholde fuldendte figurer der ikke kunne opstilles ordentligt i den egentlige rapport.

Contents

1 Indledning

Vestbjerg Byggecenter A/S har ligesom mange andre virksomheder, set fordelene i at få en digital løsning, som så mange andre virksomheder også har fået installeret. De håber, at den kan give bedre arbejdsgang og vækst, hvilket ofte er, hvad der sker når sådan en løsning indføres.

Denne rapport er udarbejdet omkring udviklingen af en del af et system til byggecenteret. Den har til opgave at belyse arbejdet og tankegangen, der er foregået under udviklingen af del-systemet.

Der vil blive lavet en analyse af virksomheden, som oplyser de vigtigste problemstillinger og ud fra dem vil systemet blive udviklet. Det vil blive belyst igennem UML diagrammer, hvordan systemet strukturers.

Part I

Forundersøgelse

2 Virksomheden

Det er virksomheden *Vestbjerg Byggecenter A/S*, som har interesse i udviklingen af et nyt IT system. De har angivet en række krav til, hvad det nye system skal have af funktionaliteter og hvilke problemstillinger de vil løse igennem det. For at klargøre kravene analyseres virksomheden først i denne forundersøgelse.

Først gennemgås en beskrivelse af organisationens struktur og ledelse, virksomhedens kultur og interessenter. Derefter en *SWOT-Analyse* og en gennemgang af workflows for virksomhedens afdelinger. Herefter afsluttes der med en *Cost-benefit* analyse.

2.1 Organisations beskrivelse og evaluering

Virksomheden er grundlagt i 1995 af Anders Olesen. I 2010 blev hans to sønner Thomas og Casper ansat i ledelsen under Anders Olesen, som er den administrerende direktør. Casper er afdelingschef i trælasthandlen og tager sig derfor af den daglige ledelse i denne afdeling. I byggemarkedet er Thomas afdelingsleder, dog er der tre ledere under ham; en for køkken- og badafdelingen og en souschef for den resterende del af byggemarkedet. Udover dem er der i byggemarkedet også kommet en ny afdeling inden for pejse og brændeovne, her er der én leder med én ekspedient under sig.

Casper Olesen er den eneste leder i trælasthandlen og har 11 medarbejdere under sig, otte af dem tager sig af kunderne der kommer i afdelingen, de resterende tre personer er ansat til at levere produkter til både private og håndværkere. Der er ansat fire personer, som tager sig af bureaukratiske opgaver, såsom registrering af bilag i finansbogholderiet og løn beregning

samt debitor- og kreditorbogholderiet.

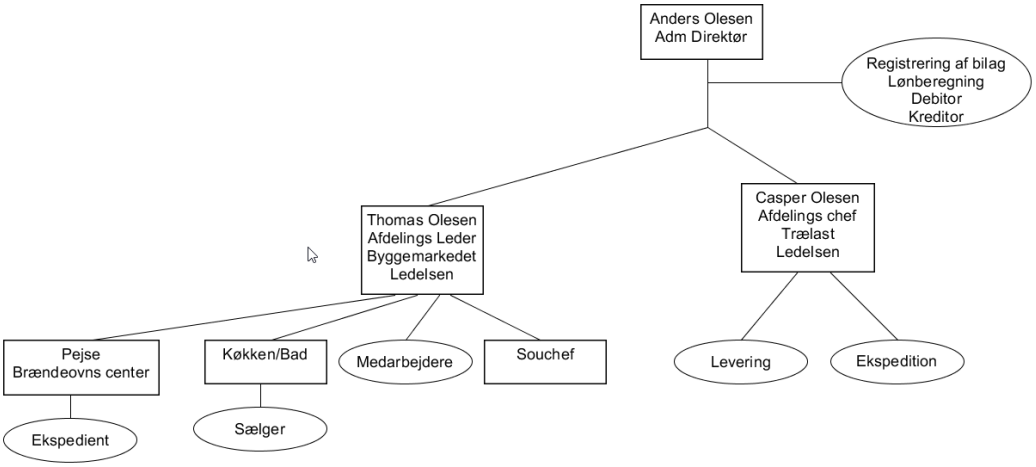


Figure 2.1: Denne figur viser organisations strukturen i Vestbjerg byggecenter A/S

Virksomheden er struktureret ud fra linjeprincippet, som giver klar ansvarsfordeling og gør, at der ikke er nogen, der modtager modstridende ordre. I figur 2.1 vil det dog kunne argumenteres, at der kan forekomme lange kommunikations veje, hvis en i trælasthandlen skal give information til en i byggemarkedet, men da det er en mindre virksomhed burde det ikke blive et problem.

Ud fra casen[4] kan det konkluderes, at virksomheden er en blanding af både mekanisk og organisk. I *The Mangement of Innovation*[3], bliver det mekaniske beskrevet som striks ledelse og klare opgavedelinger, hvor det organiske er det modsatte. Ud fra det kan der ses sammenhæng mellem det mekaniske og organiske, da virksomheden har en klar opgavedeling og interaktion mellem medarbejderne hovedsageligt er vertikal, dog efter sønnerne er kommet til, er der kommet mere fokus på de enkle medarbejdere og deres målopfyldninger.

2.2 Virksomheds kultur, ledelse og interessenter

Ud fra *casen*[4] kan det læses, at Anders Olesen har et godt forhold til alle medarbejder, men kan være lidt gammeldags i sin virksomhedsledelse og har derfor givet meget af ansvaret til sine sønner, der begge har en form for

uddannelse indenfor ledelse. Thomas lægger meget vægt på, at de enkelte medarbejder har det godt og bliver udfordret, hvor Casper er mere styrende. Overordnet er der forventinger om, at medarbejderne udfører sine opgaver, hvis dette ikke er tilfældet tager Anders Olesen en beslutning om, hvad der kommer til at ske, i værste tilfælde afskedigelse. Ledelsen bliver gennemgået videre i *SWOT-Analysen* i afsnit 2.3.

Interessenter i virksomheden består af medarbejdere, ledelse, XL-Byg og kunder.

Medarbejdere De ansatte har interesse i, at virksomheden klarer sig godt. Det vil gøre, at de har flere muligheder og kan undgå fyringer.

Ledelse Ledelsen har interesse i, at virksomheden vokser både med hensyn til produkttyper og servicer, men også økonomisk da dette vil give en større kunde tiltrækning og større omsætning.

XL-Byg Da virksomheden er en del af kæden *XL-Byg*, er der en vis interesse i, at virksomheden kører med godt overskud.

Kunder Kunder har interesse i, at virksomheden har en større omsætning da dette ville kunne give bedre priser, større og bredere udvalg.

Ud fra interessenterne kan der gives en vision til virksomheden, da det er individer, som virksomheden påvirker. De vil gerne være mere konkurrencedygtige, både i forhold til produktudvalg, men også priser.

2.3 SWOT-Analyse

I dette afsnit vil der blive beskrevet de styrker og svagheder, der kan analyseres for *Vestbjerg Byggecenter A/S*. Til at gøre dette er der foretaget en *SWOT-Analyse* til at klargøre de umiddelbare grunde til disse styrker og svagheder. *SWOT* står for *'Strength, Weakness, Opportunity, Threats'* og disse kategorier er ydere kategoriseret i interne og eksterne perspektiver.

	Hjælpelig	Skadelig
Intern	- Loyale og arbejdssomme ansatte - Klart afsat ansvar for de få ledere - Ledelsen er kuvøst anlagt og giver ansatte gode muligheder - Har haft stigende omsætning de sidste tre år	- Forældet UNIX system komplicerer informationsdeling - Udlejning af værktøj gøres manuelt - Virksomheden har ikke styr på det egentlige antal af produkter der er i besiddelse
Ekstern	- De har et stort antal af faste kunder der kan give mulighed for god omtale - Muligheder for at udvide i deres samarbejde med XL-Byg	- Mere konkurrencedygtige firmaer overtager deres potentielle forretninger - Dansk Lovgivning vedrørende produkter/materialer

Table 2.1: SWOT Tabel over Vestbjerg Byggecenter A/S

På figur 2.1 kan der ses de grunde, der er blevet kategoriseret efter firmabeskrivelsen. I øverste venstre hjørne af tabellen findes styrkerne ved firmaet. Ledelsen har en kuvøse arbejdspladskultur da Thomas Olesen og hans to sønner engagerer deres medarbejdere. Dette mindsker magtafstanden mellem dem og giver god arbejdspladskultur. Dette gør også at deres medarbejdere, der har arbejdet der i længere tid, har større vilje til at fortsætte. Derudover har der været vækst i firmaets omsætning.

Firmaets svagheder er givet i deres forældede UNIX system og de metoder de hidtil har brugt til at dele informationer og holde styr på deres produkter. De er her meget begrænsede på dette punkt, hvilket kan give problemer i et teknologisk effektiviseret samfund. Et nyt system baseret på kravene fra firmaet til konsulenterne ville løse mange af de problematikker de har.

Firmaet har et større antal af faste kunder, der kan give mulighed for god omtale i lokalsamfundet og dermed gode kunderelationer. Derudover er de integreret med *XL-Byg* som en del af kæden. Dette kan være behjælpeligt i forhold til deres navn og eventuelt udvidelse.

Som eventuelle trusler kan andre mere konkurrencedygtige firmaer overtage deres forretninger, da de har bedre styr på deres inventar og deres regnskaber på grund af bedre it-systemer end Vestbjerg Byggecenter. Det kunne anskues, at deres forretning på kort sigt ville blive mindre og mindre konkurrencedygtig, hvis ikke de får et effektiviseret IT system.

2.4 Opgaver i organisationen

I dette afsnit vil der blive beskrevet de opgaver, der er i *Vestbjerg Byggecenter A/S*, for trælast, byggemarkedet, udlejningen, genbestilling af produkter og personligt tilbud til en kunde. Dette vil blive vist igennem workflows og medarbejder-opgave-mål tabeller. Målet er at afgrænse de use cases, der findes i systemet og hvilke aktører står for hvilke. Efterfølgende laves en delkonklusion på virksomhedsanalysen.

2.4.1 Workflow

Ud fra casen[4] og korte samtaler med en repræsentant for virksomheden, er der udarbejdet workflow diagrammer. De viser arbejdsgangen og opgaverne som skal løses.

ANDREAS: skal gøres noget ved

Byggemarkedet

På figur 2.2 kan workflowet ses for Byggemarkedet. På Byggemarkedet skal det være muligt at gå ind og købe et produkt uden at skulle registreres først eller have produkter bestilt hjem. Det skal dog være muligt at blive oprettet i systemet, hvis kunden ønsker det, men det er ikke nødvendigt.

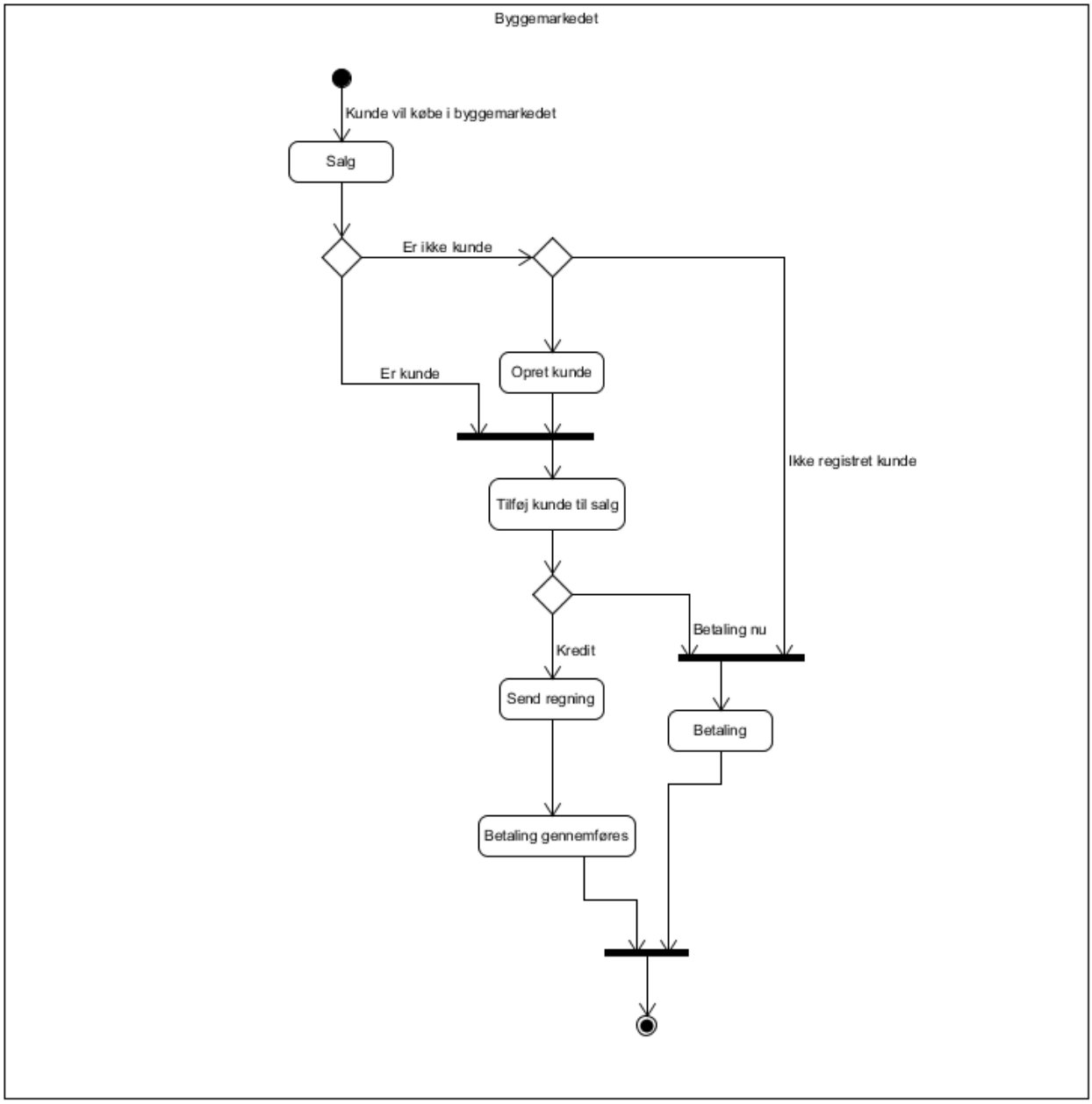


Figure 2.2: Byggemarked Workflow

Efter at en kunde, uden at registrere sig, betaler for produktet, kan de tilvælge at få produktet leveret, men de skal betale med det samme. Havde de registreret sig, ville de kunne købe produktet på kredit og få en regning tilsendt. Produktet kan derefter betales over tid indtil betalingen gennemføres og flowet terminerer.

Trælasthandlen

Ligesom Byggemarkedet kan et salg gennemføres, uden der bliver registreret, bortset fra her kan der ikke udføres et normalt køb, da produkterne er store og skal afhentes på et lager. Derfor skal der laves en ordre, når der bliver købt i trælasthandlen. Ordren betales og herefter kan det vælges at få den leveret eller afhente den selv på lageret. Det er ligesom på Byggemarkedet også muligt at købe på kredit, hvor de samme regler gælder. Workflowet terminerer når produktet er udleveret.

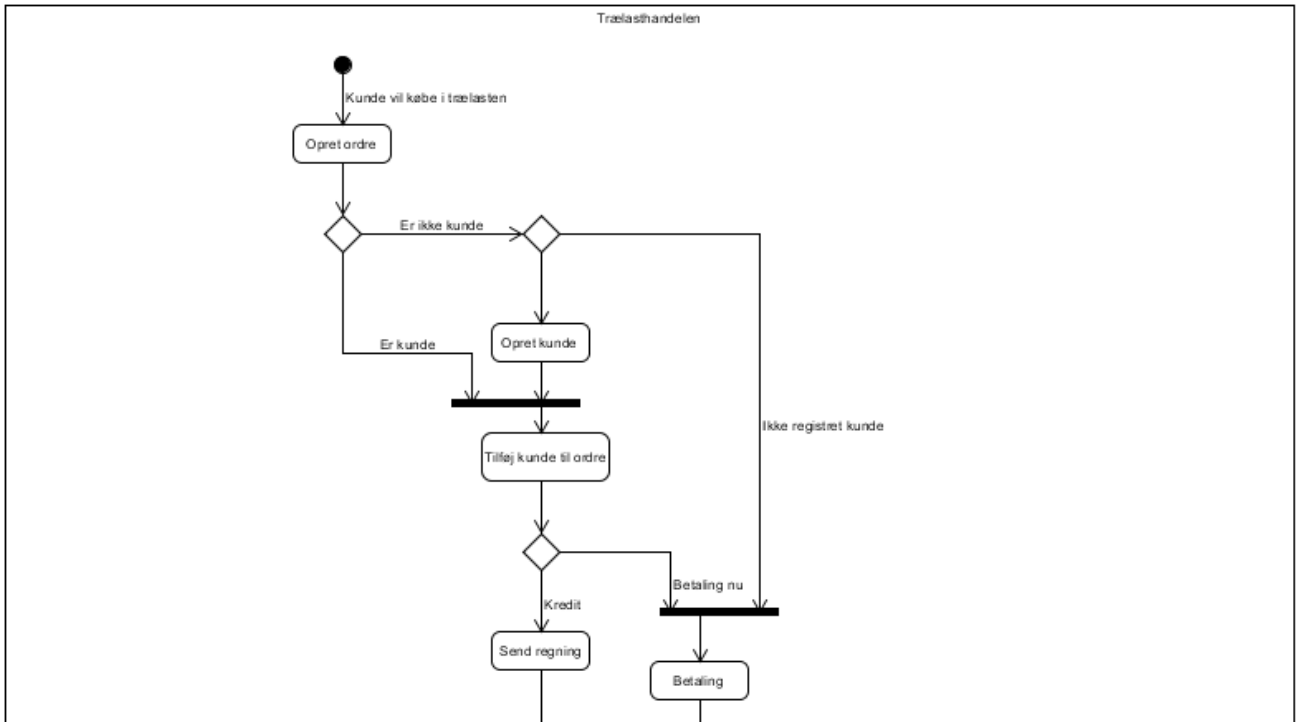


Figure 2.3: Trælasthandel Workflow 1.Del

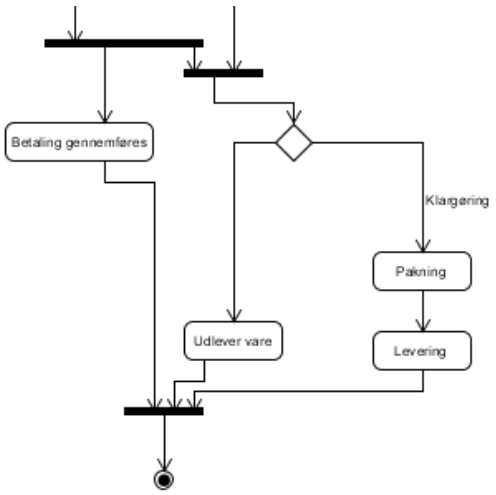


Figure 2.4: Trælasthandel Workflow 2.Del

Personligt tilbud

I virksomheden er det muligt, at sælgere kan give specielle tilbud alt efter, hvad kunden ønsker. Det skal være muligt for kunden at tage tilbuddet med hjem og have tid til at tænke over det. Det fungerer som en ordre, uden kunden får noget med hjem og uden at der sker en transaktion. I figur 2.5 er det vist, at hvis kunden ønsker tilbuddet, kan det konverteres til en ordre, ellers slettes tilbuddet og flowet slutter.

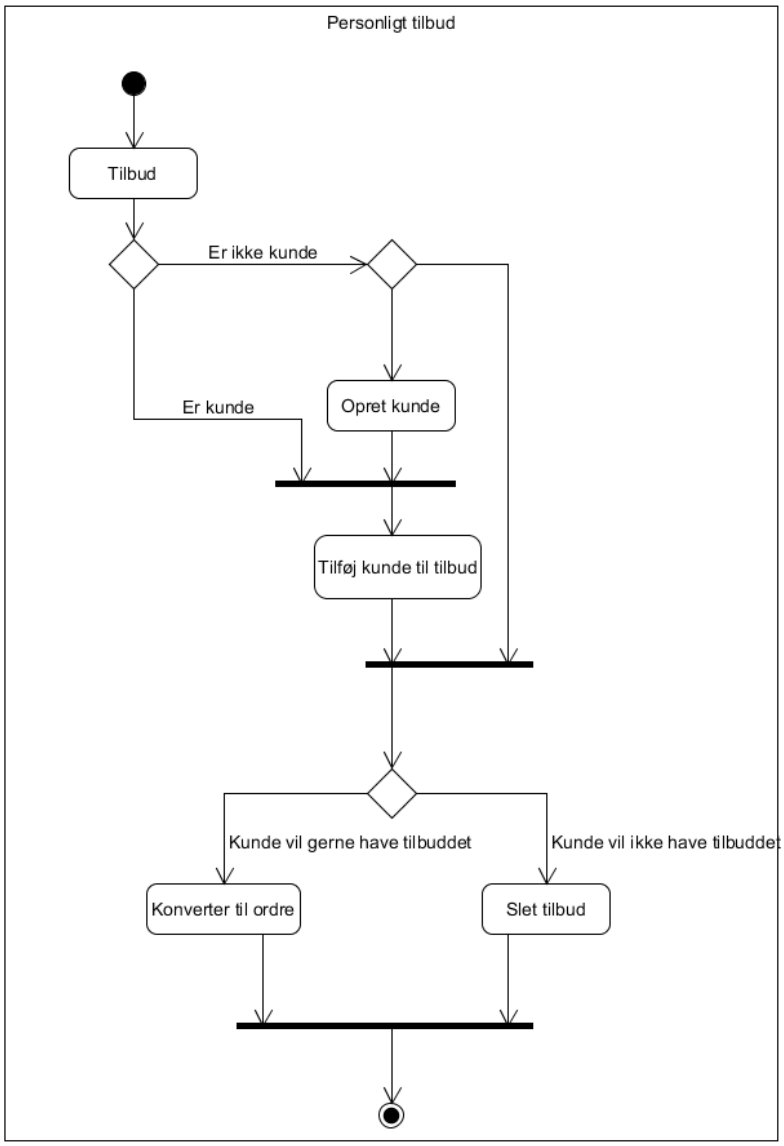


Figure 2.5: Diagram for personligt tilbud

Genbestilling

Da det skal være muligt at genbestille produkter og at det skal være nemt, vil et kommende system kunne holde styr på varebeholdningen. Derved vil det gøre det nemt at overskue, hvad der er på lageret. I figur 2.6 kan det ses, at det kun er en enkelt opgave. Opgaven skal være en form for bekræftelse, så systemet skal selv holde styr på de antal produkter, der skal genbestilles.

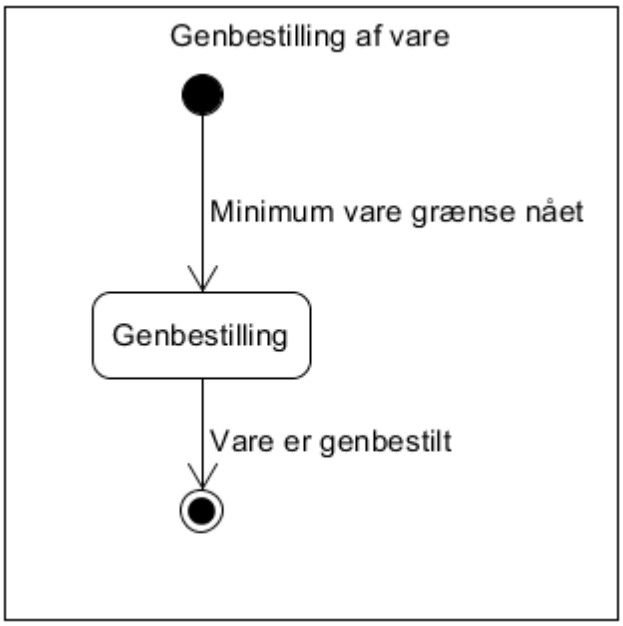


Figure 2.6: Genbestilling af produkter

Udlejning

I dette workflow, set på figur 2.7 og 2.8, er det nødvendigt, at kunden er oprettet i systemet. Grunden til dette er, at informationer omkring kunden og produktet kan findes og dermed lettere holde styr på alle udlejede produkter. En udlejning kan reserveres eller udleveres med det samme. Herefter startes udlejningsprocessen, der kan vare i et tidsrum bestemt af om produktet er reserveret af en anden kunde fremtidigt.

Kunden kan nu enten vælge at leje produktet på kredit eller indskyde et depositum indtil, de igen afleverer produktet. Når tidsrummet udløber bliver et signal sendt, at produktet skal hjemkaldes og kunden får besked. Efter produktet bliver returneret kan kunden enten betale på kredit eller direkte, hvorefter en vilkårlig tid, bliver workflowet termineret.

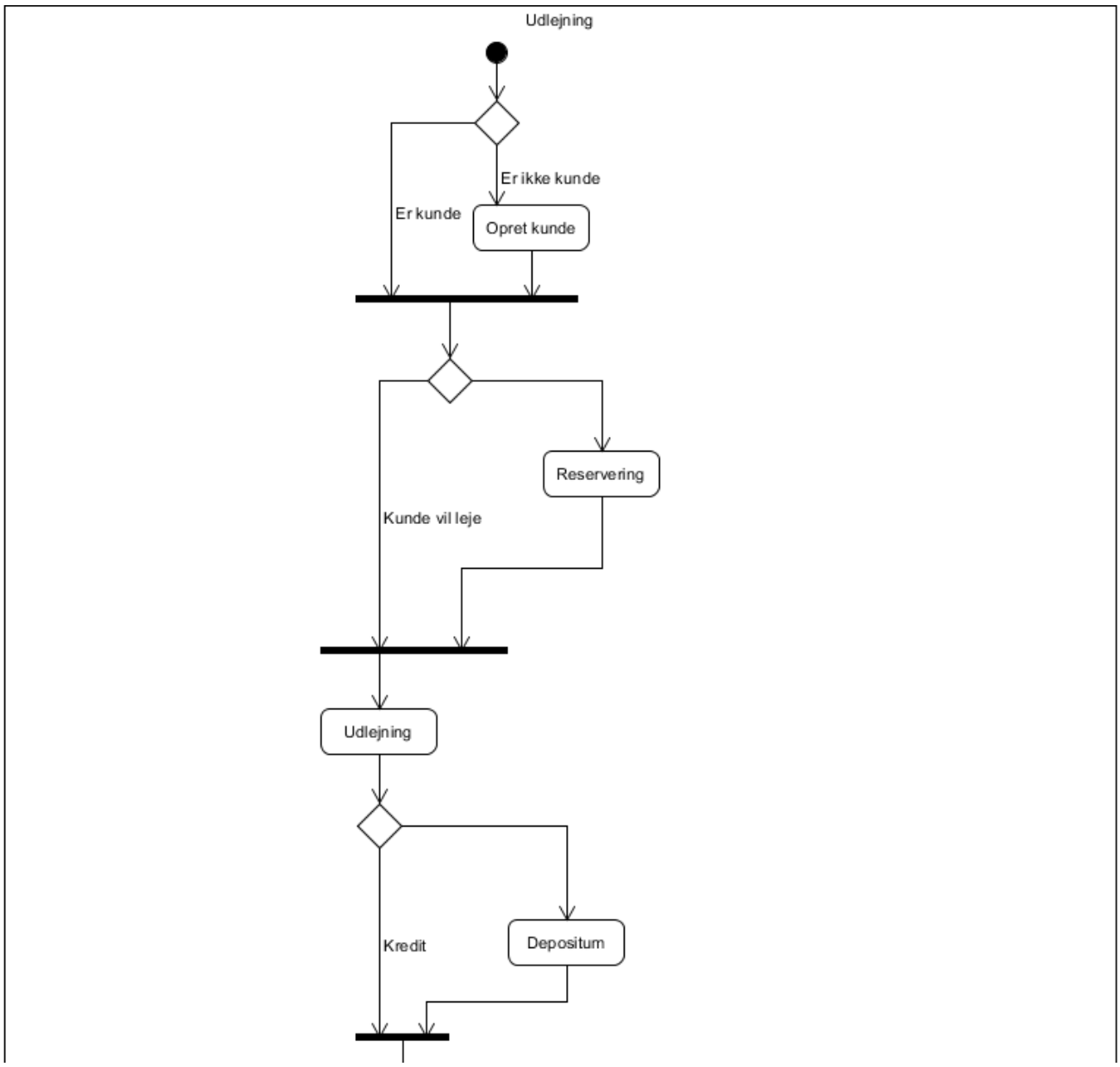


Figure 2.7: Udlejning Workflow 1.Del - Udlejning skal have en kunde registreret, først herefter kan produktet reserveres eller udlejes direkte. Et depositum eller kreditafregning påkræves.

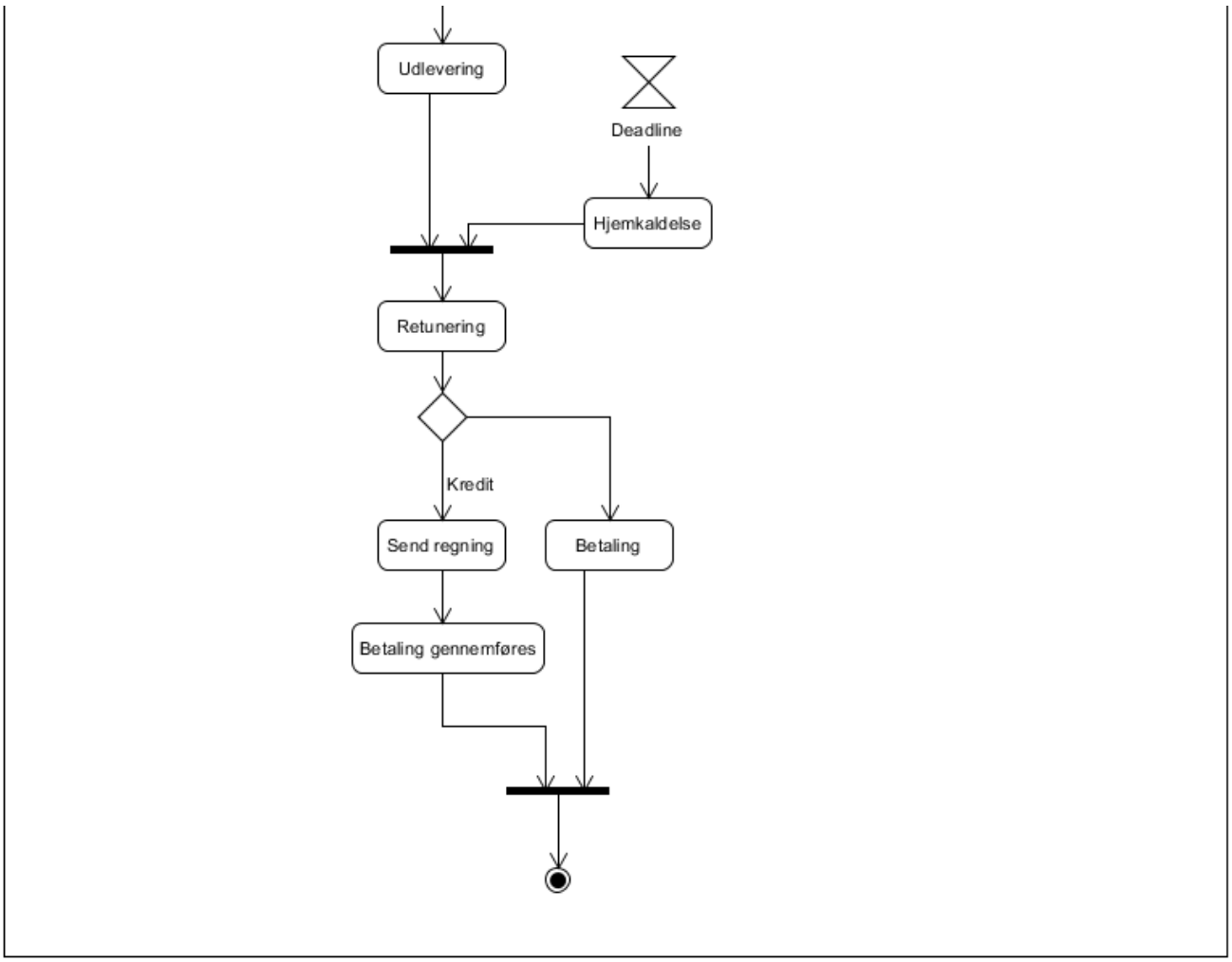


Figure 2.8: Udlejning Workflow 2.Del - Starten af udlevering af produktet indtil returnering og betalingsfasen hvorefter flowet terminerer.

I udarbejdelsen af disse Workflow er der vurderet de alternative flows, der kunne forekomme alt efter, hvad der nødvendigvis var påkrævet af de enkelte use cases. Et eksempel på dette er oprettelsen af en bruger **Opret kunde**, der på figur 2.2 for Byggemarkedet kan ses at være en ligegyldighed om en kunde er oprettet i systemet eller ej, når det kommer til et salg. Hvorimod som set på figur 2.7 for Udlejning workflowet, hvor kunden ikke kan leje noget uden at være oprettet i systemet. Dette vil sig, at i de forskellige workflows kan use cases vises, at have funktionaliteter afhængige af hinanden på forskellige måder.

2.4.2 Medarbejder-Opgave-Mål Tabeller

Tabellerne herunder er opdelt i de forskellige workflows. Det vil sige byggefirmaet på figur 2.2, trælasthandlen på figur 2.3 og på figur 2.4, genbestilling af produkter på figur 2.6, sælger tilbud på figur 2.5 og udlejningen på figur 2.7 og på figur 2.8. Nogle af opgaverne går igen i flere af tabellerne, dette betyder ikke, at det er en anden opgave, men at den kan ske flere steder.

Medarbejder	**Opgave**	**Mål**	**Step i opgave**
Ekspedient	Opret ordre	Ordren er oprettet	Tilføjer produkter
	Send regning	regning sendt	Regning bliver sendt til kunden
	Opret kunde	Kunde oprettet	Skriver oplysninger om kunden
	Tilføj kunde til ordre	kunde tilføjet	Tilføjer kunde til ordre
	Udskriv ordre	Ordreseddel	Udskriver kvittering
Lager medarbejder	Finder produkt til aflevering	produktet er fundet	Tjekker om der er flere tilbage af produktet Finder produktet frem
	Pakning	Er pakket	produkterne bliver pakket
	Lever produktet	produktet leveret	Kunden får leveret produktet Kunden får sendt produktet hjem
	Udlever produkt	produktet er udleveret	Kunden henter selv produktet

Table 2.2: Medarbejder-opgave-mål-tabel for trælast

Medarbejder	**Opgave**	**Mål**	**Step i opgave**
Ekspedient	opret tilbud	Tilbud bliver oprettet	Der bliver bestilt produkter
	Opret kunde	Kunde oprettet	Skriver oplysninger om kunden
	Tilføj kunde til tilbud Konverter tilbud til ordre Slet tilbud	kunde tilføjet tilbud konverteret tilbud bliver slettet	Tilføjer kunde til tilbud konverter tilbud til ordre slet tilbud

Table 2.3: Medarbejder-opgave-mål-tabel for tilbud

Medarbejder	**Opgave**	**Mål**	**Step i opgave**
Ekspedient	Genbestil produkter der mangler	Der bliver bestilt produkter	Bekræft at der skal genbestilles vare

Table 2.4: Medarbejder-opgave-mål-tabel for genbestilling

Medarbejder	**Opgave**	**Mål**	**Step i opgave**
Ekspedient	Opret kunde	Kunden er oprettet	Skriver oplysninger om kunden
	Udlej eller reserver værktøj	Værktøjet udlejet	Tjekker om der er flere tilbage Finder værktøj frem Tjekker tilstand af værktøj
	Udlevering	Værktøj udleveret	Værktøj bliver udleveret til kunden
	Returner værktøj	Værktøjet returneret	Værktøjet bliver returneret Tjekker tilstand af værktøj
	Send regning	Regning bliver sendt	Regning bliver sendt til kunden

Table 2.5: Medarbejder-opgave-mål-tabel for udlejning

Medarbejder	**Opgave**	**Mål**	**Step i opgave**
Ekspedient	Salg	Salg bliver gennemført	Tilføjer produkter
	Opret kunde	Kunde oprettet	Skriver oplysninger om kunden
	Tilføj kunde til salg	Kunde tilføjet	Tilføjer kunde til salg
	Sender regning	Regningen er sendt	Regningen bliver sendt til kunden

Table 2.6: Medarbejder-opgave-mål-tabel for byggemarked

2.5 Cost Benefit analyse

I dette afsnit vil der blive beskrevet de omkostninger og fordele ved det nye system. For at gøre det er der blevet lavet en Cost Benefit analyse ud fra Parker Benson analysemodellen. Modellen er opstillet efter de umiddelbare højest komplicerede use cases som set ud fra Workflows i afsnit 2.4.1, herunder **Opret Ordre**, **Udlejning**, **Salg**, **genbestilling af produkter** og **personlige tilbud**.

2.5.1 Parker Benson Analyse

Parker Benson analysen i figur 2.9 viser en oversigt over, hvordan de forskellige use cases har indflydelse på virksomheden. Totalt er værdien afgørene for beslutningen om, hvilken use case som skal laves først og/eller er vigtigst finansielt for virksomheden. Der er benyttet udleveret materiale til at gennemgå elementerne set i figuren[1]. Som set på figuren er vægtelementerne fordelt i to domæner: **Business domain** og **Technology domain**. Vægtelementerne har fået en pointgivelse fra 1-5 alt efter, hvor kraftigt de vægtes generelt set for hele systemet.

Først er der 'Return of investment' (ROI), hvilket er hvor meget virksomheden får ud af deres investering. Den er vægtet højest for **Opret Ordre** use casen, da det er primært her, at deres salg foregår igennem. Use casen **Salg** er ikke vægtet nær så høj grundet, at use casen kun bruges på Byggemarkedet, som set i afsnit 2.4.1 på figur 2.2.

Næste element er 'Strategic match' (SM) der er det højest vægtede element for alle de højest prioriterede use cases. Elementet står for hvorvidt, den udvalgte use case er i overensstemmelse med virksomhedens målsætninger[1]. Alle tre use cases har her fået samme vægtelse, da systemet gerne skal svare til, hvad virksomheden vil løse af problematikker ellers står de med et system, der ikke løser noget eller kun en del af problemerne.

	Business domain						Technology domain				
	ROI	SM	CA	MI	CR	OR	SA	DU	TU	IR	
Vægt	+3	+4	+3	+2	+1	-1	+2	-2	-1	-1	Total
Opret Ordre	12	12	2	6	1	-1	10	-6	-2	-4	30
Udlejning	3	12	2	4	1	-1	4	-4	-2	-5	14
Salg	9	12	2	6	1	-1	2	-2	-2	-2	25

Figure 2.9: Parker Benson analyse af de tre vigtigste use cases.

Næste element er 'Competitive advantage' (CA), hvilket er, hvorvidt sys-

temet vil give dem konkurrencedygtighed på markedet og om systemet giver mulighed for datatilgang, der er effektivt nok til dette formål. Den er her vægtet 3, hvilket betyder, at systemet skal give god nok datatilgang til, at virksomhedens konkurrenceevne er forbedret.

'Management information' (MI) omhandler generelt ledernes adgang til information omkring økonomi, hierarki og fordele på markedet med mere.

'Competitive response' (CR) er hvis systemet ikke gennemføres, vil det have en effekt på virksomhedens forretningsmæssige tilstand og hvordan de kan komme sig.

'Organisational risk' (OR) er vægtet lidt anderledes end andre hidtil. Den har her fået givet en negativ vægtelse. Dette er baseret på en vurdering af en række kategorier, der omhandler virksomhedens egen plan for implementeringen af det udviklede system. Kategorierne kan ses i 'Cost Benefit Analyse' materialet[1]. Den negative vægt viser, at virksomheden har en velformuleret plan for implementeringen og har ressourcerne både hos deres medarbejdere og ledere.

Herefter ses der på **Technology domain** med 'Stategic IS architechture' (SA), som kigger på, hvor fremtidsforberedt systemet er og derved hvor mange omkostninger der kunne komme i fremtiden, hvis funktionalitet skal udvides.

Herefter bliver alle værdier negativt vurderet, jo større negativ værdi jo mindre chance er der for, at projektet lykkes eller at det bliver en succes. 'Definitional uncertainty' (DU) bygger på, hvor fastlagt og kendte kravene til systemet er, både i udviklings fasen, men også hvor fastlagte brugerne er på at få et nyt system.

'Technical uncertainty' (TU) bliver bestemt ved, at der ses på virksomhedens mulighed for integrering af systemet. Desto mere der skal ændres i arbejdsmetoderne og opgaver i virksomheden, jo svære er det at integrere.

'IS infrastructure risk' (IR) her bliver det afgjort, hvor store forandringerne bliver for virksomheden som følge af projektet.

2.6 Delkonklusion

Ledelsen er kuvøst anlagt. Dette vil sige, at strukturen er en blanding af mekanisk og organisk. Ledelsen er opmærksomme på at opretholde produktive arbejdstimer, men er også nødsaglige til at give deres ansatte færre begrænsninger.

Ud fra ledelses stilen og det at virksomheden har haft en sund økonomi de seneste tre år, kan det antages, at virksomheden er velfungerende i fremgang, dog er de nuværende systemer ikke konkurrencedygtige nok i det nuværende marked. Visionen er derfor at optimere deres organisations-værktøjer, informationsdeling og lager overblik. Dette kan gøre, at de mere effektivt kan uddelegere kundevenlige beslutninger, såsom tilbud, medlemsrabatter og bredere udvalg.

Vestbjerg Byggecenter i deres samarbejde med *XL-Byg* har mulighed for at integrere deres web butik, så produkter i byggecenteret kan købes igennem denne. Dette kan også hjælpe på deres salg af produkter og reklamere tilbud og rabatter for et større antal af potentielle kunder.

I tilfælde af at virksomheden ikke får rettet op på sine svagheder, har andre virksomheder mulighed for at overtage nuværende kunder og potentielt nye kunder ellers kan der komme ny lovgivning på virksomhedens område.

Part II

Design og Implementation

3 Systemvision

Der vil i dette kapitel gennemgås visionen for systemet til at løse de problematikker, der er fundet igennem forundersøgelse af virksomheden, som bestod af en case og spørgsmål dirigeret mod virksomhedens kontaktperson. Først afgrænses formålet med systemet i hovedtræk og ud fra dette formuleres en problemstilling. Efterfølgende findes en række interessenter og brugerne af systemet som beskrives yderligere, hvorefter en række funktionelle og ikke-funktionelle krav fremlægges.

3.1 Formål og afgrænsning

Formålet er at udvikle et system, der kan hjælpe med lagerstyring, kundehåndtering og generelt at kunne hjælpe medarbejderne i hverdagen. Systemet skal understøtte aktiviteterne i ordrestyringsprocessen, fra ordren afgives, til produkterne leveres hos kunden, det vil sige registrering, pakning, levering og betaling. Igennem systemet skal det også være muligt at genbestille produkter, håndtere udlejning og oprette sælger tilbud.

3.2 Problemformulering

På nuværende tidspunkt kan det være problematisk at overskue flere ting i virksomheden, såsom kunder og især hvilke produkter der er på lageret, deres lokation og mængden af dem. Derfor ønskes et system, der kan integrere alle virksomhedens forældede systemer i et nyt system, hvor det er muligt at gøre opgaver mere brugervenlige.

3.3 Interessenter og brugere

Interessenter for systemet er som udgangspunkt de samme, som er interessenter for hele virksomheden, da systemet kommer til at påvirke hele virksomheden. Interessenterne er kunderne, lederne og medarbejderne af *Vestbjerg Byggecenter A/S* og *XL-Byg*. Medarbejderne er interessenter, da det bliver nemmere at håndtere lageret. Lederne så de kan have bedre overblik over økonomien. Kunderne så de kan spare tid. *XL-Byg* fordi de kræver en vis standard af *Vestbjerg Byggecenter A/S*.

Brugerne er lederne og medarbejderne, da det er dem, der primært kommer til at bruge systemet.

3.4 Teknologi

Teknologien er computere med internetadgang, så medarbejderne har nemmere ved at tilgå produkterne på lageret og i tilfælde af, at produktet ikke er på lager, skal de kunne bestille fra XL-Bygs hjemmeside.

3.5 Funktionalitet

- Systemet skal være brugervenligt.
- Systemet skal være pålideligt.
- Systemet skal holde styr på de forskellige produkter med forskellige priser, deres lokation og produktbeholdning.
- Systemet skal kunne udleje de forskellige udlejnings produkter.
- Systemet skal kunne holde styr på kunder og deres information.
- Systemet skal kunne lave salgs statistikker.
- Systemet skal kunne arbejde sammen med deres online butik.

4 Funktionelle krav

I dette kapitel vil der blive beskrevet de funktionelle krav. Dette vil blive vist med et use case diagram, brief beskrivelser, casual beskrivelser, fully dressed, en kandidatliste og en domænemodel.

4.1 Use case diagram

Ud fra opgaverne i virksomheden er der blevet opstillet et use case diagram, som kan ses på figur 4.1. Dette diagram viser de use cases, der skal have en indflydelse på systemet og derfor have en form for implementation. I figuren er der også kontakt med XL-byg, dette er deres netbutik, som systemet skal kunne arbejde sammen med. Der skal også kunne oprettes en ordre hos Vestbjerg Byggecenter A/S.

Figure 4.1: Use case diagram der indholder alle use cases i virksomheden.

4.2 Brief og Casual

I dette afsnit vil der blive lavet brief og casual use case beskrivelser over de forskellige use cases som er i use case diagrammet i figur 4.1, hvor de bagefter vil blive prioriteret efter arkitekturmæssig vigtighed, risici og forretnings værdi.

Opret ordre. En kunde kommer, som gerne vil købe et produkt. Medarbejderen bruger systemet til at oprette en ordre og tilføjer produktet, eventuelt tilføjer kunden til ordren, hvorefter ekspedienten gemmer ordren og printer faktura. Undervejs viser systemet informationer om produktet/produkterne, subtotal og total.

Slet ordre. En kunde fortryder et køb. Medarbejderen sletter ordren i systemet.

Opret kunde - CRUD. En kunde vil gerne oprettes i firmaets system. Medarbejderen skriver kundens oplysninger ind i systemet. Systemet viser de samlede oplysninger.

Alternativ flow. Kunden vil ikke oprettes alligevel.

Betaling. En kunde kan enten betale under købet af et produkt, eller få det på kredit, så produktet først betales senere.

Alternativ flow. Kunden har ingen penge.

Udlejning. En kunde vil gerne leje et stykke værktøj. Kunden oprettes i systemet, hvis ikke kunden er det i forvejen. Værktøjet bliver sat til at være lånt ud.

Returner produkt. En kunde er færdig med brug af leje værktøj. Værktøjet afleveres og i systemet bliver det sat klar til nyt udleje.

Alternativ flow. Kunden returnere produktet for sent.

Salg. En kunde finder et produkt. Medarbejderne scanner produktet. Lagerbeholdningen tælles ned. Produktet bliver betalt.

Reservering. En kunde skal bruge et stykke værktøj på et given tidspunkt. Systemet reserverer produktet, så dette ikke kan lejes ud til andre i perioden.

Alternativ flow. Kunden vil ikke reservere alligevel.

Levering. En kunde har købt nogle produkter og valgt at få dem sendt hjem. Systemet fortæller, at disse produkter skal pakkes sammen, hvorefter de sendes til kunden.

Udlever produkt. En kunde har købt nogle produkter og fået en faktura. Kunden bringer fakturaen til lageret, hvor kunden kan få udleveret de givne produkter.

Tilbud. En kunde skal bruge mange produkter. Kunden får derved et personligt tilbud, som kunden kan tænke over.

Genbestil produkt. Lagermedarbejderne bestiller nye produkter, da der ikke er flere af det pågældende produkt i lageret.

Opret produkt - CRUD. Firmaet får et given nyt produkt, som skal tilføjes i systemet. Ledelsen tilføjer det i systemet med de givne informationer og sætter en pris.

Alternativ flow. Produktet er allerede blevet oprettet.

Opret Ekspedient - CRUD. En Ekspedient bliver oprettet i firmaets system. Ekspedientens oplysninger bliver skrevet ind i systemet. Systemet viser de samlede oplysninger.

I tabel 4.1 ses en liste over de forskellige brief beskrivelser og hvor højt de er blevet prioriteret.

Opret ordre er blevet prioriteret som den højeste, da det som udgangspunkt er den, der har størst ROI for virksomheden, samt vurderet til at være den mest komplekse use case, derfor er det **Opret Ordre**, som vil blive udviklet først, både når det kommer til system design og implementation. Det bør dog bemærkes, at udlejning har mange af de samme problemstillinger, som opret ordre og derfor har den samme kompleksitet, men en lavere ROI. Dette kan også ses i afsnit 2.5 for Parker-Benson analysen.

Use case	**Prioritet**	**Nummer**
Opret ordre	Høj	1
Udlejning	Høj	2
Salg	Høj	3
Betaling	Medium	4
Udlever produkt	Medium	5
Returner	Medium	6
Genbestil produkt	Medium	7
Slet ordre	Medium	8
Tilbud	Medium	9
Reservering	Medium	10
Levering	Lav	11
Opret produkt - CRUD	Lav	12
Opret ekspedient - CRUD	Lav	13
Opret kunde - CRUD	Lav	14

Table 4.1: Prioriterings liste

4.3 Fully dressed

I dette afsnit vil der blive lavet en fully dressed figur 4.2 over den valgte og højst prioriteret use case, som her er **Opret Ordre**.

Use case navn	Opre ordre	
Aktører	Ekspedient	
Præ	Der eksisterer mindst et produkt i systemet	
Post	Ordre er oprettet	
Frekvens	100-300 per uge	
Flow of events	**Aktør**	**System**
	1. En kunde vil købe et produkt i trælasten	
	2. Ekspedient opretter en ordre	3. Systemet beder om en medarbjer
	4. Ekspedient vil gerne tilføje sig selv til ordre	5. Systemet beder om en kunde
	6. Ekspedient finder kunden i systemet	7. Systemet beder om et produkt
	8. Ekspedienten finder et produkt	9. Systemet returnere produktet
	10. Ekspedienten tilføjer et antal af produktet til ordren	11. Systemet til produktet med antal til ordren
	12. Ekspedienten afslutter ordren	13. Systemet opretter ordren
Alternative Flows	6a. I stedet for ordren bliver afsluttet. 1. Kunden har ikke en konto, men vil gerne oprettes. 2. Ekspedienten udfylder kundeoprettelse formular. 3. Systemet gemmer den oprettede kunde. 4. Ekspedienten opretter kunden til ordren. - Flowet genoptages herefter fra punkt nr. 7. 12a. Kunden vil ikke købe alligevel 1. Ekspedienten anullerer ordren	

Figure 4.2: Fully Dressed over opret Ordre

4.4 Domænemodel

I dette afsnit vil der blive beskrevet en domænemodel, som kan ses på figur 4.3, hvor der første laves en kandidat liste, som kan ses i figur 4.2, den er lavet over de mulige klasser og nogle af de attributter, der skal være i domænemodellen. Kandidaterne bliver kort evalueret om de er nødvendige eller unødvendige ud fra det, bliver de tilføjet til domænemodellen. I figuren 4.2 er Leader og Rank ekskluderet, dette skyldes, at de har lidt den samme funktion, hvor rank kunne være en attribut der har Leader som mulighed.

Candidates for classes	**Evaluation**	**In/Out**
Order	Order is needed to sell products of a certain type	Inclusive
OrderLine	Is used to control the amount of each product making an order	Inclusive
Employee	Contains information about a employee	Inclusive
Offer	Is used to create special offers for customers	Inclusive
Delivery	Is used to keep track of the delivery for each order	Inclusive
Payment	Saves information about an order and its payment details	Inclusive
Customer	Contains information about a customer	Inclusive
Product	Contains all the information about products	Inclusive
ProductList	A container for all products	Exclusive
ProductHandling	This functionality should be on the product or the product list	Exclusive
Rental	Controls which product(s) are rented out to a customer	Inclusive
Person	Superclass that derive employee and customer	Inclusive
Rank	Might as well be an attribute on employee	Exclusive
Leader	Might as well be an attribute on employee	Exclusive
Sale	Superclass of payment which control sales that does not need a registred customer	Inclusive
ProductPackage	A product that is a collection of chosen products	Inclusive
OfferLine	Controls number of products on an Offer	Inclusive
Item	Superclass to product and productPackage	Inclusive
DiscountType	Contains all kinds of discount a customer can have	Inclusive

Table 4.2: Kandidater til domænemodellen

På figur 4.3 kan der ses klasser, der er markeret med rød. De er blevet markeret sådan fordi de er blevet overvejet som en del af systemet, men er ikke blevet implementeret. De ikke-markerede er dem som er essentielle for **Opret Ordre**.

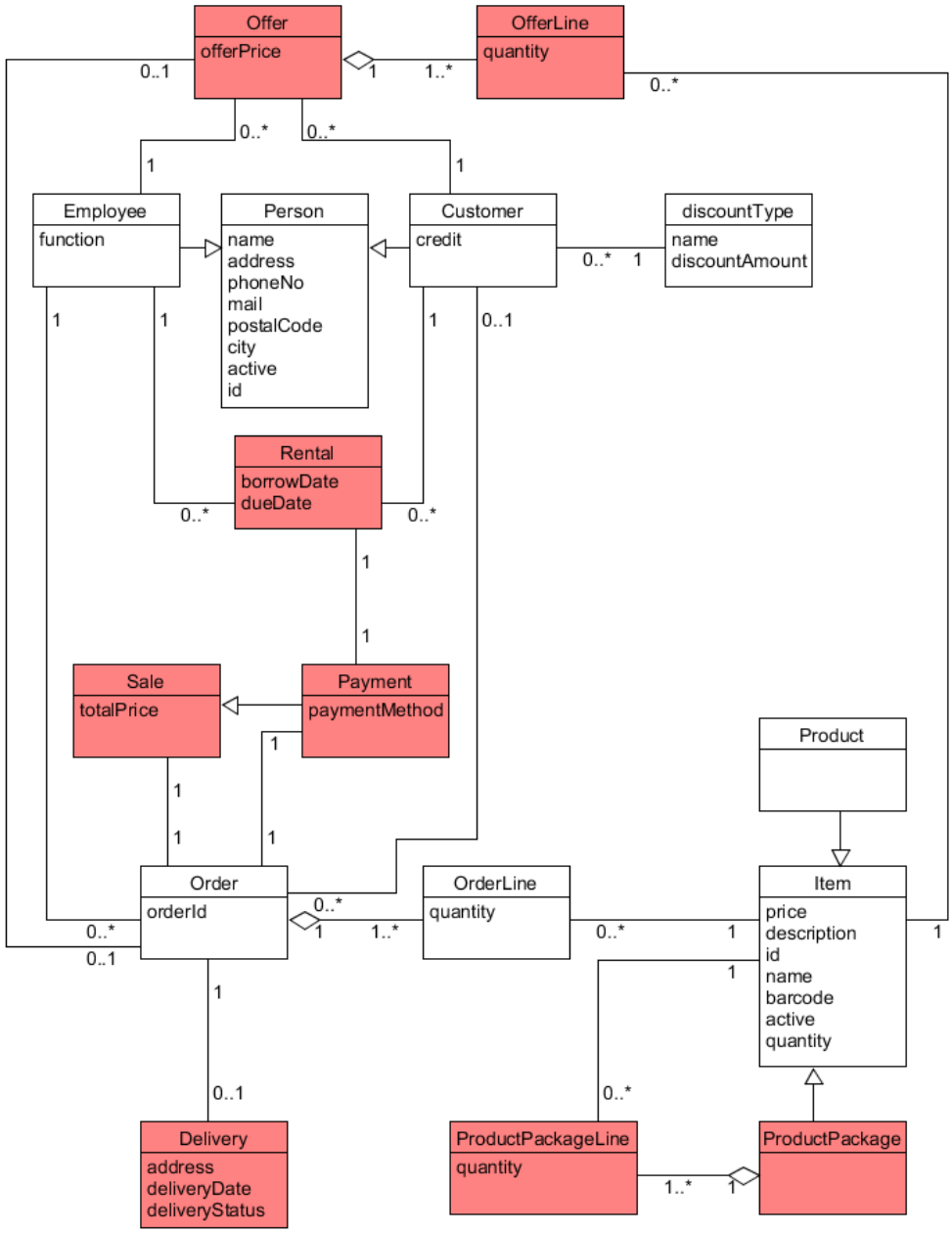


Figure 4.3: Domæne model

4.5 Delkonklusion

Ud fra de use case der blev fundet i kapitel 2 blev deres flow beskrevet igennem brief og casual use case beskrivelser. Sammen med beskrivelserne og Parker Benson analysen i afsnit 2.5.1 er der blevet lavet en prioriterings liste, hvor det blev bekræftet at **Opret Ordre** er den use case, der fremad rettet vil blive arbejdet med. Det er derfor der er lavet en fully dressed beskrivelse af opret ordre, som viser use casens flow i dybden.

I figur 4.3 bliver klasse kandidaterne i tabel 4.2 visualiseret og de mest relevante for Opret Ordre er markeret med hvidt. Så efter dette kapitel er den mest komplekse use case fundet.

5 Analyse

I dette kapitel vil analysen af systemet blive beskrevet. Analysen bliver beskrevet med et system sekvens diagram og operations kontrakter.

5.1 System sekvens diagram og operations kontrakter

I dette afsnit vil der blive beskrevet system sekvens diagram for **opret ordre** og de tilhørerende operations kontrakter. Disse kan ses på figur 5.1.

For use casen **Opret Ordre** er der blevet lavet et system sekvens diagram, der starter med at oprette en ordre. For at en ordre kan oprettes tilføjes der en medarbejder, en kunde, hvor begge af disse bliver fundet med deres telefonnummerer, herefter bliver der fundet produkter og disse blive tilføjet til ordren.

Ud fra system sekvens diagrammet er der blevet lavet operations kontrakter over `createOrder` og `addItemToOrder`, da de laver ændringer i systemet. Derfor er `findEmployeeByPhoneNo`, `findCustomerByPhoneNo` og `findItem` blevet undladt, da disse ikke laver ændringer i systemet.

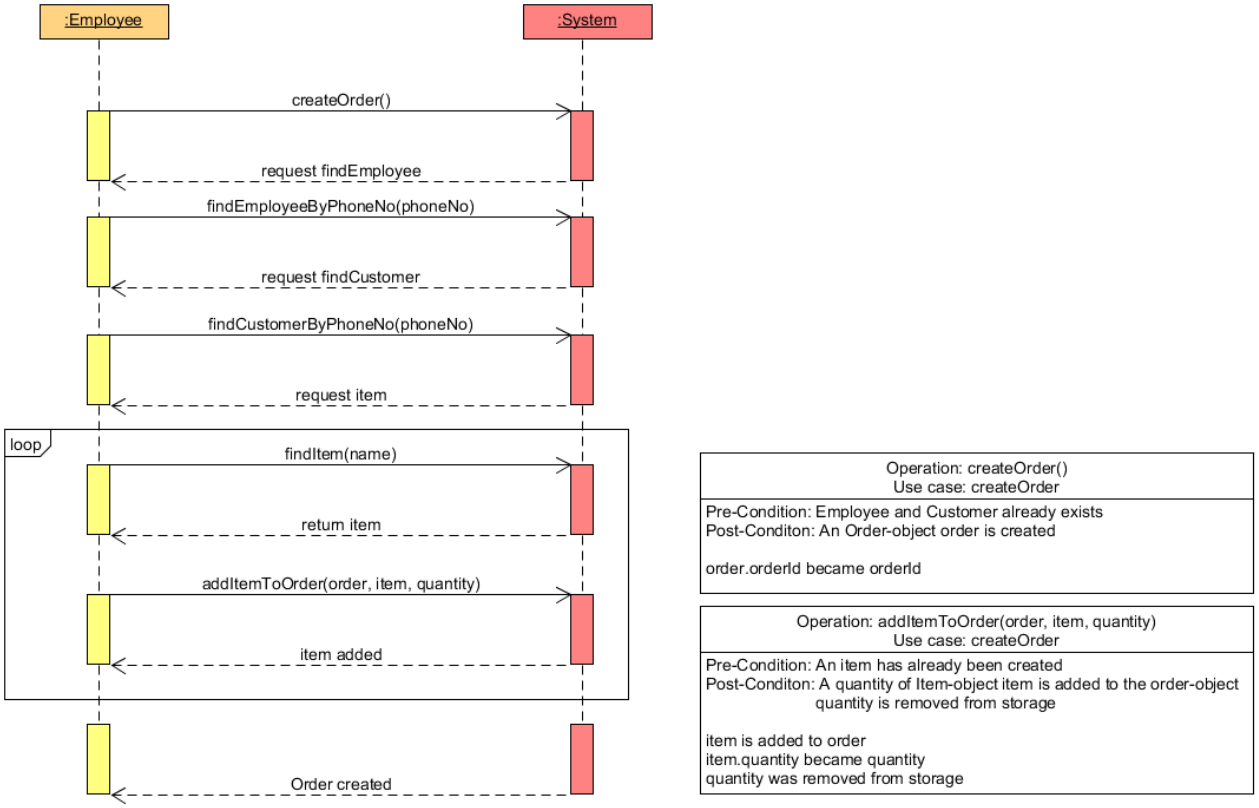


Figure 5.1: System sekvens diagram og operations kontrakt over opret Ordre

Ud fra denne figur kan der nu konstrueres et kommunikations- og design klasse diagram der omhandler opret ordre. Sammen med figur 4.2(fully dressed) giver det også indblik i hvordan en succesfuld ordre oprettelse skal fungere systemrelateret.

6 Design

I dette kapitel vil der blive beskrevet design af systemet. Dette bliver beskrevet med et design klasse diagram og et interaktionsdiagram.

6.1 Design klasse diagram

I dette afsnit vil der blive lavet et design klasse diagram over systemet. Det samlede design klasse diagram kan findes i bilag A.2.

6.1.1 Employee

Her ses en del af design klasse diagrammet for Employee, som ses på figur 6.1. Det der ses er en `EmployeeUI`, en `EmployeeController`, en `EmployeeContainer`, en `Employee` og en superklasse for `Employee` som hedder `Person`. Det kan også ses, at der er åben tre lags struktur, da UI laget og controller laget kan læse de informationer, der er i model laget. Det kan ses, at der er åben tre lags struktur da `EmployeeUI` og `EmployeeController` kan læse de informationer der er i `Employee`.

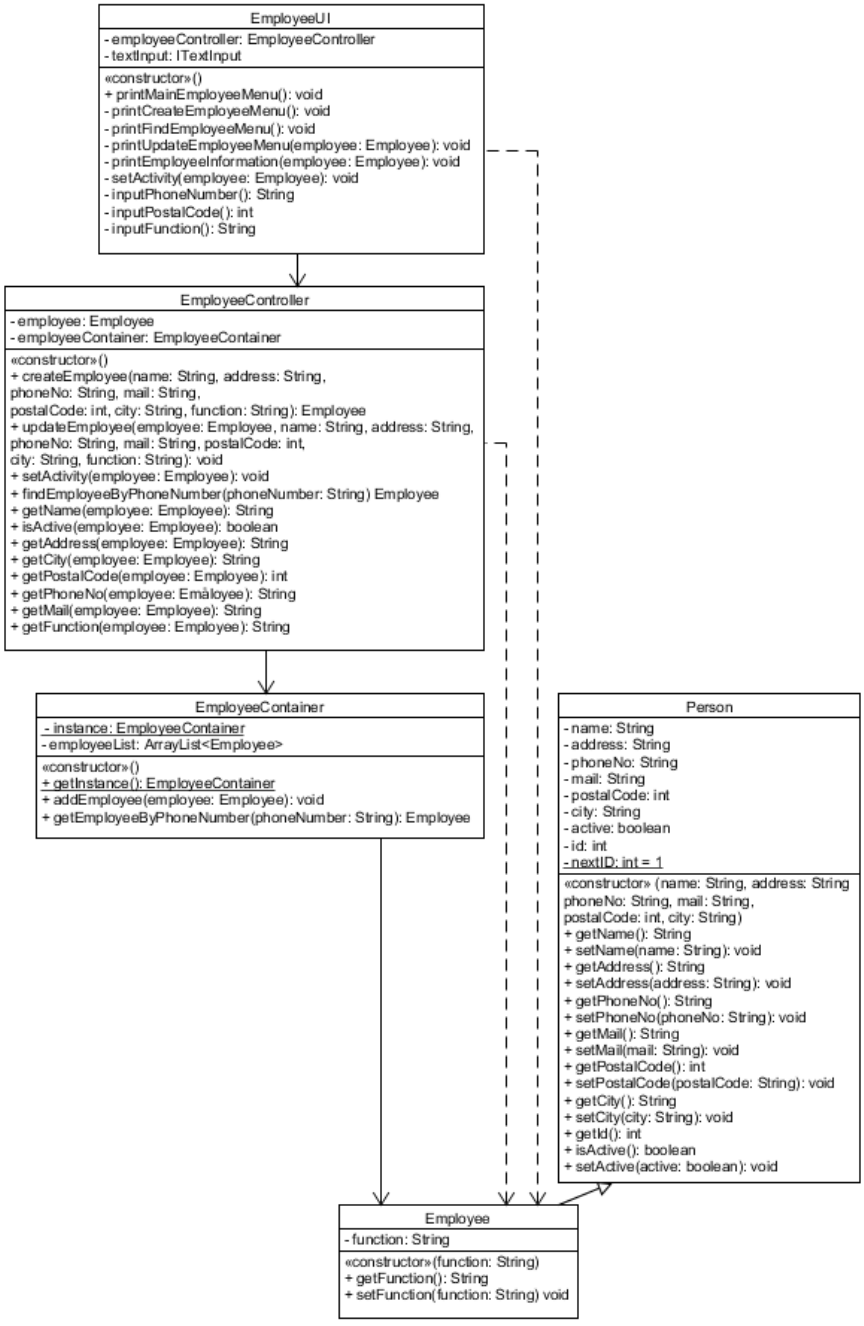


Figure 6.1: Design klasse diagram for Employee

6.1.2 Customer

Her ses en del af design klasse diagrammet for Customer, som ses på figur 6.2. Det der ses er en `CustomerUI`, en `CustomerController`, en `CustomerContainer`, en `Customer` og en superklasse for `Customer` som hedder `Person`. Det kan også ses, at der er åben tre lags struktur, da UI laget og controller laget kan læse de informationer, der er i model laget. Det kan ses, at der er åben tre lags struktur, da `CustomerUI` kan læse de informationer der er i `Customer` og `CustomerController` kan læse de informationer, der er i `Customer` og har direkte forbindelse til `DiscuontType`.

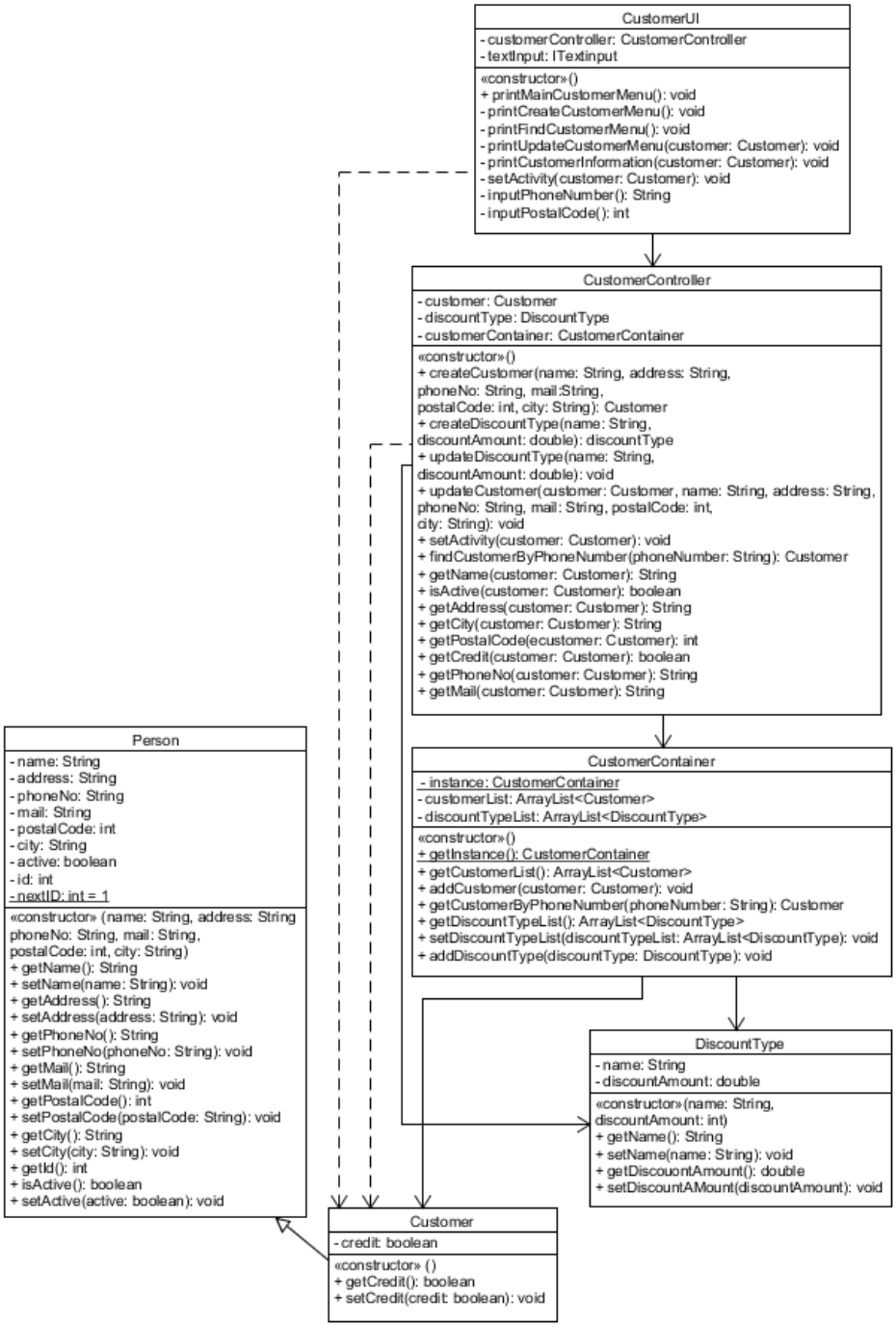


Figure 6.2: Design klasse diagram for Customer

6.1.3 Order

Her ses en del af design klasse diagrammet for Order, som ses på figur 6.3. Det der ses er en `OrderUI`, en `OrderController`, en `OrderContainer` og en `Order`. Det kan også ses, at der er åben tre lags struktur, da UI laget og controller laget kan læse de informationer, der er i model laget. Det kan ses, at der er åben tre lags struktur, fordi `OrderUI` kan læse de informationer der er i `Order` og `OrderLine` og `OrderController` laget kan læse de informationer der er i `Order`.

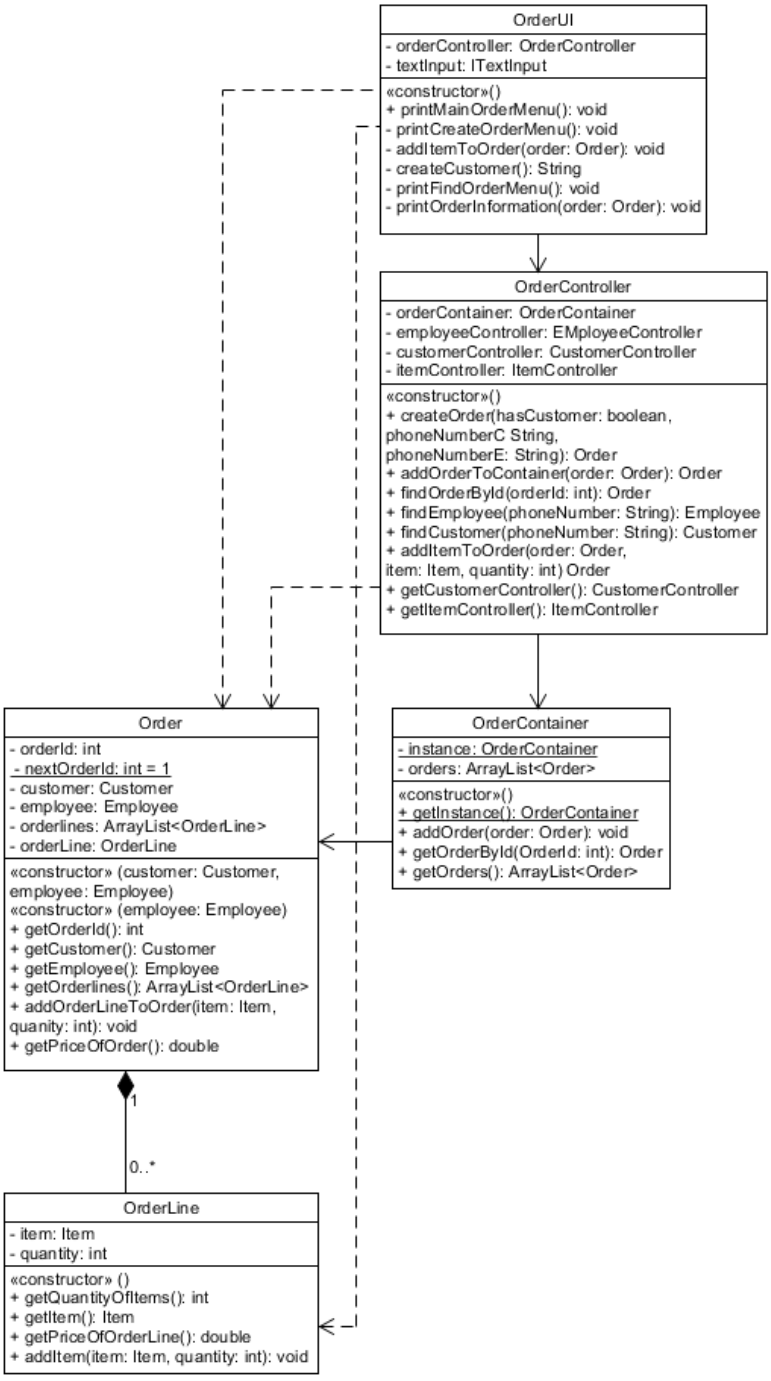


Figure 6.3: Design klasse diagram for Order

6.1.4 Item

Her ses en del af design klasse diagrammet for Item, som ses på figur 6.4. Det der ses er en `ItemUI`, en `ItemController`, en `ItemContainer` og `Item` som er superklasse for `Product` og `ProductPackage`. `ProductPackage` og `ProductPackageLine` er blevet markeret med røde, fordi de i start blev implementeret, men blev dog ikke lavet færdig og de er ikke blevet integreret i systemet. Det kan også ses, at der er åben tre lags struktur, da UI laget og controller laget kan læse de informationer, der er i model laget. Det kan ses, at der er åben tre lags struktur fordi `ItemUI` kan læse de informationer, der er i `Item` og `ItemController` kan læse de informationer der er i `Item` og `Product`.

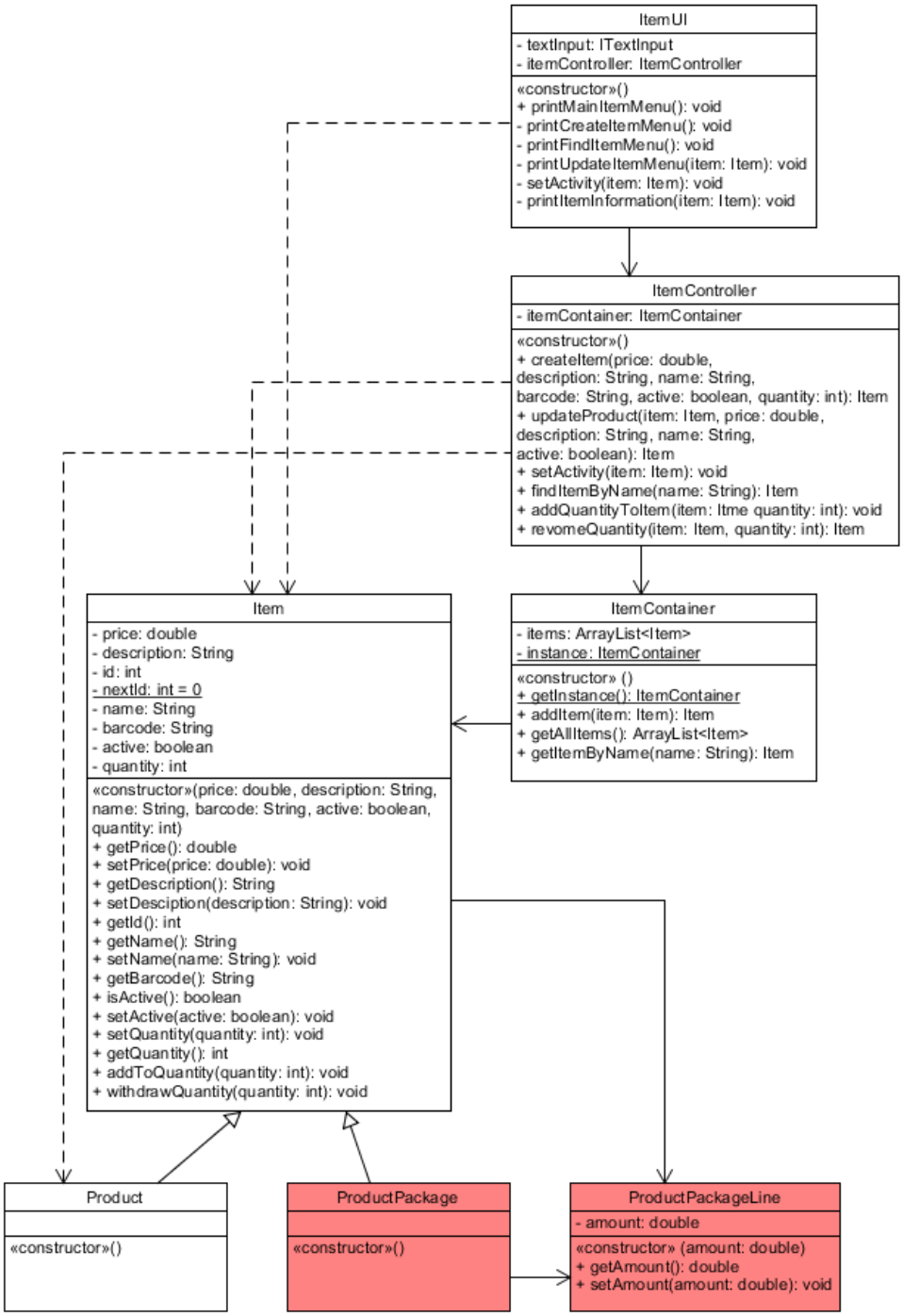


Figure 6.4: Design klasse diagram for Item

6.2 Interaktionsdiagram

I dette afsnit vil der blive beskrevet et interaktionsdiagram. Der er to forskellige interaktionsdiagrammer. I dette projekt er der blevet valgt at lave et kommunikationsdiagram, som her kan ses på figur 6.5. Kommunikationsgiagrammet er lavet for use casen **opret Ordre**.

I figur 6.5 er det som tidligere nævnt opret ordre, der bliver lavet et flow gennemgang af. Her skulle det være muligt at starte fra metoden `1. order = createOrder()`, følge tallende op til metoden `4.2 addOrder(order)` som tilføjer `Ordre` objektet til containeren.

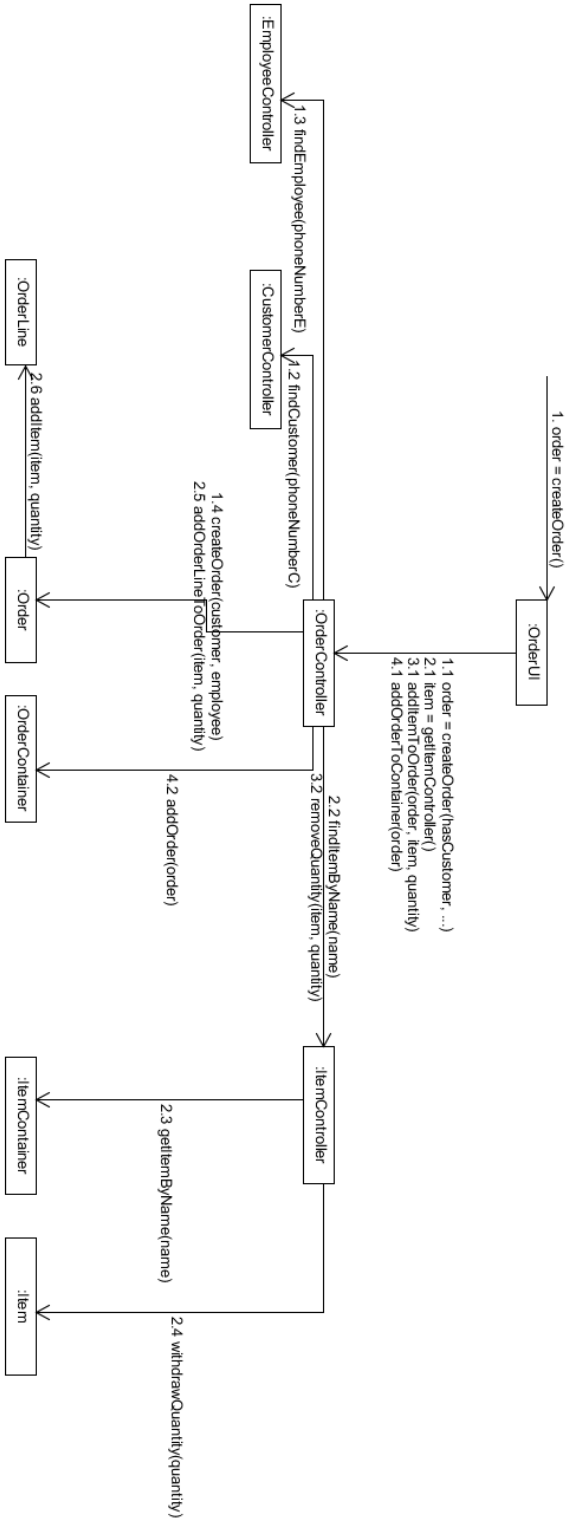


Figure 6.5: Kommunikationsdiagram over opret ordre.

6.3 Delkonklusion

Igennem dette kapitel blev domæne modellen, SSD og operations kontrakterne brugt til at udforme et kommunikations diagram og et design klasse diagram.

Kommunikations diagrammet viser, hvilke metodekald som skal laves for, at use casen kan gennemføres i systemet. I design klasse diagrammet bliver klasserne til alle de tidligere nævnte metoder specificeret sammen med deres hjælpe metoder, såsom get og set metoder. Med disse to diagrammer færdig kan der nu implementeres et system med basis ordre funktionalitet.

7 Implementation

I dette kapitel vil der blive gennemgået implementeringen af koden, kode konventioner og system struktren. Der vil blive lagt vægt på implementationen af use casen **Opret Ordre**, dette vil blive vist med kode udsnit hvor relevant.

7.1 Kode konventioner

Systemets source kode er bygget op ud fra nogle konventioner, som gør opstillingen af koden konsistent, hvilket vil sige, at det holder sig til en bestemt måde at skrive diverse variabel navne og så videre, disse er beskrevet herunder.

Som udgangspunkt bruges der Sun Microsystems Inc. Java kode konventioner[2], hvis dette ikke er tilfældet beskrives det herefter.

- I tilfælde af parametere af for eksempel `String` typen vil parameteren skrives som `metodeNavn(String string)`, hvis det er en primitiv type gives der et beskrivende navn, eksempelvis `name`.
- Variabler skal være så korte som muligt imens de stadigvæk er beskrivende, det samme er gældende for metoder.
- I tilfælde af for-each loops skal objektet der ligger i kollektionen navngives “element”.
- I tilfælde af at en metode skal returnere, kaldes `return` kun en gang og derfor laves en lokal variable med navnet "res" af returnerings typen.

7.2 System arkitekturen

Den logiske arkitektur bygger på at diverse klasser bliver opdelt i pakker. Pakkerne bliver lavet alt efter, hvilke lag der skal være i systemet og hvor pakkerne hører til. De fleste systemer har tre lag, et **UI lag**, der tager imod input fra brugeren og sender disse input ned til laget nedenunder, nemlig **kontrol laget**.

I kontrol laget bliver brugerens input fra det forrige lag brugt til at finde information og lave ændringer i systemet. Heriblandt oprettelsen, opdateringer og andet manipulering af objekter. Informationerne der bliver fundet kommer fra det nederste lag, som er **model laget**. Det er også i model laget at ændringer i systemet bliver foretaget.

I denne trelags struktur kan det vælges at have en striks lag arkitektur, hvilket betyder, at lagene kun kan kommunikere med de nærmeste lag. Dette system bruger dog en mere afslappet lag arkitektur, som giver muligheden for, at UI laget kan få information fra model laget, dog må UI laget ikke lave ændringer i model laget uden at gå igennem kontrol laget. Hvilket også betyder, at UI laget ikke direkte ændrer noget, men det er kontrol laget der gør det.

Denne struktur burde give et overskueligt system, som senere kan tilføje ekstra funktionalitet eller ændre eksisterende funktionalitet uden at være den originale udvikler. Dette sikres ved at opstille arkitekturen så det er indlysende nok, hvad hvert lag kan og ikke kan.

7.3 System beskrivelse

I de følgende afsnit vil der blive beskrevet de tre lag med hensyn til, hvad de internt har ansvar for og specifikt hvilke klasser de interagerer med.

7.3.1 Brugergrænseflade

Her gennemgås hvilke klasser er implementeret i `uiLayer` pakken. Brugergrænsefladen i dette projekt er et tekstbaseret og udnytter et bibliotek som er udviklet til præcist dette brug. Fra biblioteket bliver metoden `promptIntBetween(String, String, int, int)` brugt mest. Metoden giver udvikleren mulighed for at printe en besked, hvor der kan svares med tal mellem et minimum og et maksimum og hvis brugeren giver et ugyldigt input, får brugeren en fejlbesked.

Brugeren kan derfor give input til grænsefladen, som har metoder, der benytter deres respektive *Controller* i laget under den. Brugergrænsefladen har derfor ikke mulighed for at manipulere information direkte, kun aflæsning og kommunikation af information til kontrol laget.

MainUI Heri ligger `main()` metoden, hvorfra programmet starter. Metoden kalder `printMainMenu()`, der er startsiden for programmets tekst interface.

CustomerUI Klassen indeholder metoder der er nødvendige for at håndtere funktionaliteter for kunder. Den benytter `CustomerController` klassen til dette.

EmployeeUI Klassen indeholder lignende metoder som fra `CustomerUI`, her med henblik på håndteringen af ansatte. Den benytter `EmployeeController` klassen til dette.

ItemUI Klassen indeholder metoder, der håndterer produkterne i systemet, heri oprettelsen og muligheden for at se informationer omkring et specifikt produkt. Den benytter `ItemController` til disse formål.

OrderUI Klassen styrer oprettelsen og informationer relateret til ordrer. Dette gør den igennem `OrderController` klassen. Klassen vil beskrives dybere i afsnit 7.4.

7.3.2 Kontrol laget

Her gennemgås hvilke klasser er implementeret i `controlLayer` pakken. Kontrol laget håndterer klasserne i model laget og modtager information fra brugeren i brugergrænsefladen. Disse informationer er nødvendige for kontrol lagets funktionaliteter.

CustomerController Håndterer `Customer`, `CustomerContainer` og `DiscountType` klasserne. Den kan eksempelvis oprette kunder og ligge dem i containeren. Klassen modtager information fra `CustomerUI` af brugeren.

EmployeeController Klassen har samme funktionaliteter som `CustomerController`, men med håndteringen af ansatte igennem `Employee` og `EmployeeContainer` klasserne. Klassen modtager information fra `EmployeeUI` af brugeren.

ItemController Håndterer klasserne `Item` og `ItemContainer`. Klassen kan oprette produkter, der er nødvendige for oprettelsen af en ordre, hvilket `OrderController` gør brug af. Metoderne i klassen modtager informationer fra brugeren igennem `ItemUI`.

OrderController Håndterer klasserne `Order`, `OrderLine` og `OrderContainer`. Den står for oprettelsen af ordrer, applikationen af produkter og hvilke persontyper skal være en del af ordren (kunder og ansatte). Klassen modtager information fra brugeren igennem grænsefladeklassen `OrderUI`. Klassen vil beskrives dybere i afsnit7.4.

7.3.3 Model laget

Her gennemgås hvilke klasser er implementeret i `modelLayer` pakken. Model laget indeholder klasser, der har til formål at være instanstierbare, så de kan benyttes af kontrol laget til oprettelsen af objekter. Klasser såsom `Customer` og `Employee` har ikke funktionalitet der rækker ud over 'set-' og 'get' metoder. Bortset fra `Order` klassen der har nogle få andre funktionaliteter. Disse beskrives herunder.

Kunder i model laget

Alle klasser der har med kunder at gøre i model laget er beskrevet herunder.

Customer Nødvendig for oprettelsen af en kunde. Klassen arver fra superklassen `Person`. Hver person har en række nødvendige informationer som både `Employee` og `Customer` gør brug af. Forskellen mellem de to er, at `Customer` har en `credit` variabel for at afgøre, om en kunde er på kredit. Dette er implementeret, men ikke integreret i systemet.

DiscountType Klassen giver mulighed for at oprette forskellige tilbudstyper. Denne funktionalitet er implementeret, men er ikke integreret i systemet.

CustomerContainer Denne klasse er en singleton og kan indeholde instanstierede `Customer` og `DiscountType` objekter. Containeren indeholder metoder, der gør den i stand til at finde specifikke kunder, der ligger i dens liste og alle deres informationer. Den kan ikke oprette kunder eller discount typer.

Ansatte i model laget

Herunder beskrives alle klasser der involverer ansatte i systemet.

Employee Nødvendig for oprettelsen af en ansat. Klassen arver også fra superklassen `Person`. Forskellen mellem `Customer` og `Employee` er, at ansatte har en `function` variabel, der bestemmer, hvilken stilling den ansatte har, men er ikke integreret.

EmployeeContainer Indeholder instanstierede `Employee` objekter. Denne klasse gør det også muligt at finde specifikke ansatte i dens liste.

Produkter i model laget

Herunder beskrives alle klasser, der involverer produkter i systemet.

Item Klassen er en superklasse, som `Product` og `ProductPackage` klasserne arver fra. Klassen giver blandt andet information omkring pris, om den er aktiv i systemet og antal. Antallet er givet i variablen `quantity` og er vigtig for oprettelsen af en ordre, da den skal bruge information om, hvor mange der findes af dem i systemet.

Product Klassen er ikke anderledes rent indholdsmæssigt fra `Item`. Den er ikke direkte benyttet til at identificere og oprette produkter, her er `Item` brugt i dets sted. Dette er gjort for at gøre det muligt at oprette produkter og produktpakker (`ProductPackage`) igennem `Item` instanstieringer, da begge underklasser til `Item` har præcist samme instansvariabler.

ProductPackage Denne klasse er implementeret, men ikke integreret i systemet. Den adskiller sig fra `Product`, da den er en kollektion af produkter, der bliver til dets eget produkt. Til dette indeholder den en liste af produkter den kan tilføje produkter og andre produkt pakker til.

Ordre i model laget

Herunder beskrives klasserne der involverer oprettelsen og varetagning af ordre i systemet. De beskrives selektivt i afsnit 7.4.

Order Klassen har instansvariabler af persontyperne i model laget og `OrderLine` klassen. Klassen har metoder, der sørger for at tilføje orderlines til ordren. En `Order` er en kollektion af orderlines.

OrderLine Klassen indeholder instansvariabler, der indeholder et `Item` og et antal. En `OrderLine` kan tilføje et og kun et produkt til sig og et antal, hvilket `Order` benytter sig af til at udregne pris for alle sammenlagte orderlines.

OrderContainer Klassen indeholder alle oprettede ordre i systemet. Den har metoder til at finde informationer omkring specifikke ordre. Dette gør brugergrænsefladen sig nyttig af til at fremvise.

7.4 Implementation af Opret Ordre

I dette afsnit vil der blive gået i dybden med implementation og integration af de nødvendige klasser og metoder til at gøre use casen **Opret Ordre** mulig for brugeren at gennemføre. Afsnittet er struktureret efter tre-lags arkitekturen.

7.4.1 OrderUI

Denne klasse er som beskrevet tidligere ansvarlig for kommunikation mellem bruger og klassens underliggende controller. Klassen er opdelt i en række metoder, der hver styrer en menu på grænsefladen.

Disse menu metoder består af `printMainOrderMenu()`, `printCreateOrderMenu()` og `printFindOrderMenu()`. `printMainOrderMenu()` er hovedmenuen for ordre brugergrænsefladen og giver adgang til alle funktionaliteterne med hensyn til håndtering af ordrer.

Til oprettelsen af en ordre kaldes `printCreateOrderMenu()` fra den tidligere nævnte menu. Metoden kan ses på figur 7.1. Metoden gør brug af to lokale variabler for henholdsvist kunden og den ansattes telefonnummer. Metoden spørger brugeren, om de vil oprette en ordre uden en kunde er sat på, hvilket er en kunde, der er oprettet førhen i systemet.

```
private void printCreateOrderMenu() {

    int choice;

    String customerPhone;
    String employeePhone;
    Order order = null;

    System.out.println("[1] Create Order with a Customer.");
    System.out.println("[2] Create Order without a Customer.");
    System.out.println("[0] Cancel.");

    choice = textInput.promptIntBetween("Input valid choice", "That is not a valid input", 0, 2);
    employeePhone = textInput.promptString("Please input Employee Phone No.");

    switch (choice) {
    case 1:
        customerPhone = textInput.promptString("Please input Customer Phone No.");

        if (orderController.findCustomer(customerPhone) == null) {
            customerPhone = createCustomer();
        }
        order = orderController.createOrder(true, customerPhone, employeePhone);
        break;
    case 2:
        order = orderController.createOrder(false, null, employeePhone);
        break;
    case 0:
        break;
    default:
        break;
    }
    addItemToOrder(order);
}
```

Figure 7.1: Kode udsnit af printCreateOrderMenu() metoden. Bemærk addItemToOrder() metoden der kaldes til sidst.

Hvis der vælges at oprette en ordre med en kunde tilvalgt, skal der også indskrive kundens telefonnummer, dog hvis personen ikke eksisterer i systemet bliver `createCustomer()` kaldt så de kan oprettes. Denne metode eksisterer i denne klasse og gør brug af `OrderController`, der kommunikerer med `CustomerController`, der har en metode til at oprette en kunde. Metoden returnerer det indtastede telefonnummer, da personen blev oprettet. Herefter bliver ordren oprettet med kundens og den ansattes navn tilvalgt.

`addItemToOrder()` metoden i klassen kan ses på figur 7.2. Metoden spørger brugeren, om de vil tilføje et produkt, der findes i systemet til ordren. Når de vælger at gøre dette, bliver brugeren spurgt om navnet på produktet og antallet. Hvis antallet overskrider det, der er på lageret, starter processen igen.

Produktet bliver tilføjet til ordren, hvis der var nok produkter i systemet.

Herefter spørges brugeren igen om de vil tilføje et produkt, hvis nej, så bliver den nu oprettede ordre tilføjet til `OrderContainer` og udskriver informationen omkring transaktionen og oprettelsen.

```
private void addItemToOrder(Order order) {
    boolean running = true;
    int choice2;
    Item item;

    while (running) {
        System.out.println("Do you want to add a Product?");
        System.out.println("[1] Yes.");
        System.out.println("[2] No, continue.");
        choice2 = textInput.promptIntBetween("", "That is not a valid input", 1, 2);

        switch (choice2) {
        case 1:
            String name = textInput.promptString("Input a Name on a Product.");
            item = orderController.getItemController().findItemByName(name);
            if (item != null) {
                int quantity = textInput.promptInt("Input quantity of the chosen Product", "Input a valid number");
                if(item.getQuantity() >= quantity) {
                    orderController.addItemToOrder(order, item, quantity);
                }else {
                    System.out.println("There are only " + item.getQuantity() + " " + item.getName() + " left");
                }
            } else {
                System.out.println("Couldn't find that item.");
            }
            break;
        case 2:
            running = false;
            break;
        }
    }
    orderController.addOrderToContainer(order);
    printOrderInformation(order);

}
```

Figure 7.2: Kode udsnit af addItemToOrder().

7.4.2 OrderController

Denne controller klasse har adgang til de andre controllere i systemet, hvilket er nødvendigt for at indsamle de korrekte informationer omkring kunder, ansatte og produkter.

Klassen håndterer oprettelse-, opbevaring af ordrer og tilføjelse af produkter til ordrer. To mærkbare metoder er `createOrder()`, som set på figur 7.3 og `addItemToOrder()`, som set på figur 7.4.

```
public Order createOrder(boolean hasCustomer, String phoneNumberC, String phoneNumberE) {
    Order order = null;
    if(hasCustomer == true) {
        order = new Order(findCustomer(phoneNumberC), findEmployee(phoneNumberE));
    } else {
        order = new Order(findEmployee(phoneNumberE));
    }
    return order;
}
```

Figure 7.3: Kode udsnit af createOrder() i controller klassen.

Som gennemgået i afsnit 7.4.1 vedrørende de to valg som brugeren kunne tage ved oprettelsen af en ordre, med eller uden en oprettet kunde. Denne metode kigger på den parametiserede boolean værdi, der er i metode headeren og gør brug af `Order` klassens to constructors, hvori den ene tager imod en kunde og den anden gør ikke. Baseret på valget returneres en `Order`.

```
public Order addItemToOrder(Order order, Item item, int quantity) {
    Item res;
    res = itemController.removeQuantity(item, quantity);
    if(res != null) {
        order.addOrderLineToOrder(res, quantity);
    }
    return order;
}
```

Figure 7.4: Kode udsnit af addItemToOrder() i controller klassen.

Metoden `addItemToOrder()` har parametiseret en `Order` og en `Item` samt et antal. Hvis antallet ikke overskrider, hvor mange af det produkt der er parametiseret, så oprettes en ny `OrderLine` igennem metoden `addOrderLineToOrder()`, der kaldes igennem den parametiserede `Order`. Denne metode gennemgås i afsnit 7.4.3.

7.4.3 Opret Order i model laget

Herunder gennemgås de klasser og metoder, der benyttes til oprettelsen af en ordre. I klassen `Order` findes den førnævnte metode `addOrderLineToOrder()`, der er parametiseret med et produkt og et antal. Kode udsnittet kan ses på figur 7.5.

```
public void addOrderLineToOrder(Item item, int quantity) {
    orderLine = new OrderLine();
    orderLine.addItem(item, quantity);
    orderLines.add(orderLine);
}
```

Figure 7.5: Kode udsnit af addOrderLineToOrder() i Order klassen.

Når denne metode kaldes i controlleren oprettes en `OrderLine` på det produkt, der blev parametiseret. Herefter tilføjes den instanstierede `OrderLine` til en liste af orderlines. I brugergrænsefladen er det muligt at blive ved med at tilføje produkter til ordren, hvilket her betyder, at der oprettes en `OrderLine` hver gang tilhørende hvert produkt. Dette medfører til sidst at en ordre består af en række orderlines, som prisen kan udregnes fra.

7.5 Test af system

Test af systemet blev som udgangspunkt gjort på klasser i kontrol laget, kort efter klasserne var færdige. Til testene blev *JUnit 4* brugt til at verificere om variabler var det de skulle være efter diverse metoder var eksekveret.

I figur 7.6 kan en JUnit testcase ses, denne bliver brugt til at konkludere, om det er muligt at oprette en ordre. I den er der blevet instanstieret et `Employee` objekt, et `Customer` objekt, et `Ordre` objekt med deres telefonnummere, der binder dem til ordren, to `Item` objekter som får tilføjet en `quantity`, som her er 1 på det ene objekt og 2 på det andet. Dette bliver gjort for at kunne konkludere, om det er muligt at tilføje en varieret `quantity` og at det bliver fjernet fra "lageret".

Det sidste bliver verificeret igennem en `assertEquals(int expected, int actual)` metode, som tager to integer, en for hvad der forventes og en for den der skal verificeres. Der bliver også brugt en `assertEquals(double expected, double actual, int decimals)` til at bekræfte, at produkter bliver tilføjet til ordren. Denne tager dog en double som i dette tilfælde skal være 196.67, da det er den samlede pris på ordren. På denne måde bliver ordrelinjens vigtigste funktion også testet, nemlig udregningen på en ordrelinje.

```
@Test
public void testAddItemToOrderAndCalculatePrice() {
    EmployeeController employeeController = new EmployeeController();
    CustomerController customerController = new CustomerController();

    employeeController.createEmployee("Jacob", "Revlevej", "90909090", "Høj@gmail.com", 9000, "Aalborg",
            "Medarbejder");
    customerController.createCustomer("Joan", "Govej", "12345678", "moj@gmail.com", 9020, "IngeBy");

    OrderController orderController = new OrderController();
    Order order = orderController.createOrder(true, "12345678", "90909090");

    ItemController itemController = new ItemController();
    Item boremaskine = itemController.createItem(90.5, "Boremaskine til at bore med", "Boremaskine", "9823", true, 100);
    Item tandpasta = itemController.createItem(15.67, "Tandpasta til at børste med", "Tandpasta", "9983", true, 100);

    itemController.addQuantityToItem(tandpasta, 10);
    itemController.addQuantityToItem(boremaskine, 10);

    orderController.addItemToOrder(order, tandpasta, 1);
    orderController.addItemToOrder(order, boremaskine, 2);

    assertEquals(109, itemController.findItemByName("Tandpasta").getQuantity());
    assertEquals(108, itemController.findItemByName("Boremaskine").getQuantity());

    assertEquals(196.67, order.getPriceOfOrder(), 2);
}
```

Figure 7.6: Kode udsnit af en Junit 4 testcase

De resterende testcases der er blevet lavet af systemet, er bygget op på en meget lignende måde. Hver testcase skal køres for sig selv, da de alle instanstiere de objekter de skal bruge. Hver case bliver eksekveret for sig selv fordi der arbejderes med Singletons og der ikke er implementeret en metode til at slette referencer fra den originale `arrayList`, så en `tearDown()` metode ville ikke nulstille test miljøet. I alt er der blevet lavet test af en af containerne og to af controllerne. Disse blev valgt da de havde en form for kompleksitet og containerne er lavet på samme måde gennem hele systemet.

Part III

Konklusion og Evaluering

8 Konklusion

Den udarbejdede rapport formår at komme ind på virksomheden, dens interne forhold, arbejdsopgaver og krav. Igennem en prioriteringsliste og en Parker Benson analyse, der kan ses som meget overfladisk, da den er blevet udarbejdet af system udviklere og ikke virksomheden selv. Derved blev fokuset lagt på **Opret ordre** som er den mest komplekse use case. På denne case bliver der udarbejdet en grundig beskrivelse igennem flere forskellige diagrammer, der skal vise, hvordan casen kan forløbe, både lavpraktisk og teknisk. Ud fra diagrammerne, der er udarbejdet, er der blevet implementeret en del af systemet der skulle kunne håndtere hele virksomheden.

Delen af systemet omfatter det mest basale i forhold til at angive en ordre for en kunde, samt udregne ordrens pris, men nogle af metoderne i systemet kunne dog godt have mere funktionalitet. Derudover mangler ordren diverse attributter, der kan håndtere dato, tilbud, kunde rabat og så videre, derfor vil systemet ikke kunne bruges til dagligt, som det er nu. Systemet kan stadigvæk ikke lave arbejde sammen med en online butik, lave salgstatestikker eller udleje. Dette kunne dog integreres i det nuværende system når eller hvis der skulle udarbejdes i det fulde system.

9 Evaluering

9.1 Proces og gruppe evaluering

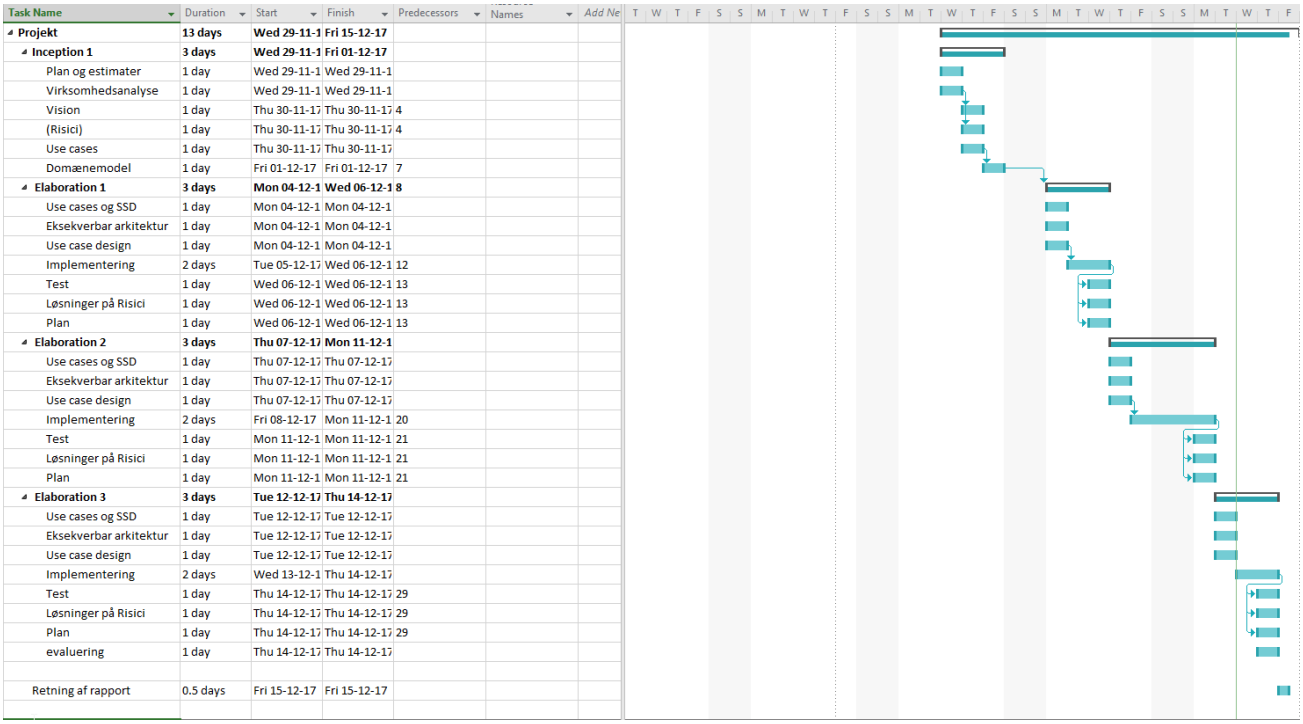
Processen af projektet har overordnet gået godt. Vi har dog over kompliceret noget af arbejdet i forhold til, hvad og hvor meget der skulle være med i systemet. Da vi begyndte at skrive koden havde vi **productPackage** og **productPackageLine**, men senere indså vi, at det var irrelevant at have med i forhold til den nødvendige funktionalitet til systemet, så her blev der lavet unødvendig arbejde. Alt i alt, blev der fokuseret for meget på funktionalitet, som ikke var nødvendigt for at der kunne oprettes en ordre.

Vi har været gode til at diskutere på en respekterende måde og har været gode til at sørge for at alle har haft muligheden for at arbejde med alle opgaver, såsom når det kom til system implementation, da vi sad to og to sammen og programmerede. Vi syntes, at dette fungerede godt. På grund af dette fik vi også diskuteret tingene mere. Dette har været godt for læringsprocessen. Dog blev effektiviteten ikke så god, da det gik langsommere med at få skrevet koden. Der har også været en positiv atmosfære i gruppen, da gruppe medlemmerne har været gode til at overholde gruppekontrakten.

9.2 Evaluering af planlægning

Planlægningen af projektet blev gjort ved hjælp af Microsoft Project Professional (MPP), som er et udbredt projekt styring program samt ved hjælp af Trello, som er et online opgave styrings program for teams og grupper. I Trello blev opgaver lavet, som vi kom frem til i projektet. I MPP lavede vi et gant-kort, som kan ses på figur 9.1. På Gant-kortet kan det ses, at pro-

jektet er delt op i 4 iterationer, dog mangler milepæle. Inception fasen har en iteration, hvor planen var at beskrive virksomheden og finde den første use case der skulle implementeres. Denne fase gik som planlagt.



Task Name	Duration	Start	Finish	Predecessors	Names
Projekt	**13 days**	**Wed 29-11-1**	**Fri 15-12-17**		
Inception 1	**3 days**	**Wed 29-11-1**	**Fri 01-12-17**		
Plan og estimater	1 day	Wed 29-11-1	Wed 29-11-1		
Virksomhedsanalyse	1 day	Wed 29-11-1	Wed 29-11-1		
Vision	1 day	Thu 30-11-17	Thu 30-11-17	4	
(Risici)	1 day	Thu 30-11-17	Thu 30-11-17	4	
Use cases	1 day	Thu 30-11-17	Thu 30-11-17		
Domænemodel	1 day	Fri 01-12-17	Fri 01-12-17	7	
Elaboration 1	**3 days**	**Mon 04-12-1**	**Wed 06-12-1**	8	
Use cases og SSD	1 day	Mon 04-12-1	Mon 04-12-1		
Eksekverbar arkitektur	1 day	Mon 04-12-1	Mon 04-12-1		
Use case design	1 day	Mon 04-12-1	Mon 04-12-1		
Implementering	2 days	Tue 05-12-17	Wed 06-12-1	12	
Test	1 day	Wed 06-12-1	Wed 06-12-1	13	
Løsninger på Risici	1 day	Wed 06-12-1	Wed 06-12-1	13	
Plan	1 day	Wed 06-12-1	Wed 06-12-1	13	
Elaboration 2	**3 days**	**Thu 07-12-17**	**Mon 11-12-1**		
Use cases og SSD	1 day	Thu 07-12-17	Thu 07-12-17		
Eksekverbar arkitektur	1 day	Thu 07-12-17	Thu 07-12-17		
Use case design	1 day	Thu 07-12-17	Thu 07-12-17		
Implementering	2 days	Fri 08-12-17	Mon 11-12-1	20	
Test	1 day	Mon 11-12-1	Mon 11-12-1	21	
Løsninger på Risici	1 day	Mon 11-12-1	Mon 11-12-1	21	
Plan	1 day	Mon 11-12-1	Mon 11-12-1	21	
Elaboration 3	**3 days**	**Tue 12-12-17**	**Thu 14-12-17**		
Use cases og SSD	1 day	Tue 12-12-17	Tue 12-12-17		
Eksekverbar arkitektur	1 day	Tue 12-12-17	Tue 12-12-17		
Use case design	1 day	Tue 12-12-17	Tue 12-12-17		
Implementering	2 days	Wed 13-12-1	Thu 14-12-17		
Test	1 day	Thu 14-12-17	Thu 14-12-17	29	
Løsninger på Risici	1 day	Thu 14-12-17	Thu 14-12-17	29	
Plan	1 day	Thu 14-12-17	Thu 14-12-17	29	
evaluering	1 day	Thu 14-12-17	Thu 14-12-17		
Retning af rapport	0.5 days	Fri 15-12-17	Fri 15-12-17		

Figure 9.1: Gant-kort over tidsplan på projektet

I elaboration fasen var der 3 iterationer. Her gik den første også som planlagt, dog blev der ikke lavet "løsning på risici" og "plan", da det virkede irrelevant og da tidsplanen blev lavet, var det ikke indlysende, hvad de skulle bruges til.

I elaboration 2 gik vi igennem det vi havde lavet i elaboration 1 og byggede videre på det. Denne iteration blev der dog fokuseret mere på programmering end, hvad tidsplanen viser, endelig i elaboration 3 blev tidsplanen delvist ignoreret og i Trello blev der oprettet opgaver, som vi manglede og det var dem vi hovedsageligt arbejdede efter. Vi havde også overset tiden, der skulle bruges på dokumentation af systemet og var ikke regnet med i tidsplanen. Planlægningen har delvist haft en positiv effekt på projektet, men forståelsen for udviklingen af den var ikke tilstrækkelig og derved mindskende det produktiviteten og brugen af den.

Bibliography

[1] Cost Benefit Analyse. ucn.instructure.com, 2017. [Online, Handed Material, accessed 04-12-2017].

[2] Sun Microsystems Inc. Java Code Coventions. http://www.oracle.com/technetwork/java/codeconventions-150003.pdf, 1997. [Online, accessed 01-12-2017].

[3] G.M. Stalker Tom Burns. *The Management of Innovation*. Oxford University Press, 1994.

[4] UCN Datamatiker. *Vestbjerg Byggecenter A/S*, 2017.

Ordliste

Interressenter Er personer som har en eller anden form for interesse i systemet.

SWOT-Analyse Strength, Weakness, Opportunity and Threats som er en analyse, der går ind og analyserer virksomheden på disse områder.

Cost-Benefit analyse er en analyse, som går ind og kigger på prisen kontra det, som man får ud af et system.

Linjeprincippet er en ledelses form, som er karakteriseret ved, at den enkelte medarbejder har én og kun én direkte overordnet, samt at arbejdsopgave og ansvar er klart fordelt.

UNIX system er et udbredt styresystem til computere, servere og arbejdsstationer

Singleton er en instans af en container, som kun kan oprettes en gang, hvorefter det kan bruges gentagne gange.

A

Appendix

A.1 Gruppe kontrakt

- Alle møder til den aftalte tid 8:30 - 16:00, hvis man bliver forsinket eller ikke kommer, giver man besked hurtigst muligt.
- Hvis der gives hjemmearbejdet for i gruppen, laves det til den aftalte tid.
- Hvis man bliver forhindret eller har problemer med hjemmearbejdet, forventes det, at man giver besked til gruppen.
- Alle skal deltage aktivt i gruppearbejdet samt diskussioner.
- Rapporten bliver skrevet i LaTex.
- Systemet bliver implementeret vha. eclipse.
- Der bliver fuldt javas kode konventioner
- Når der implementeres, bliver det gjort i kode grupper af mindst 2 personer.

A.2 Design klasse diagram

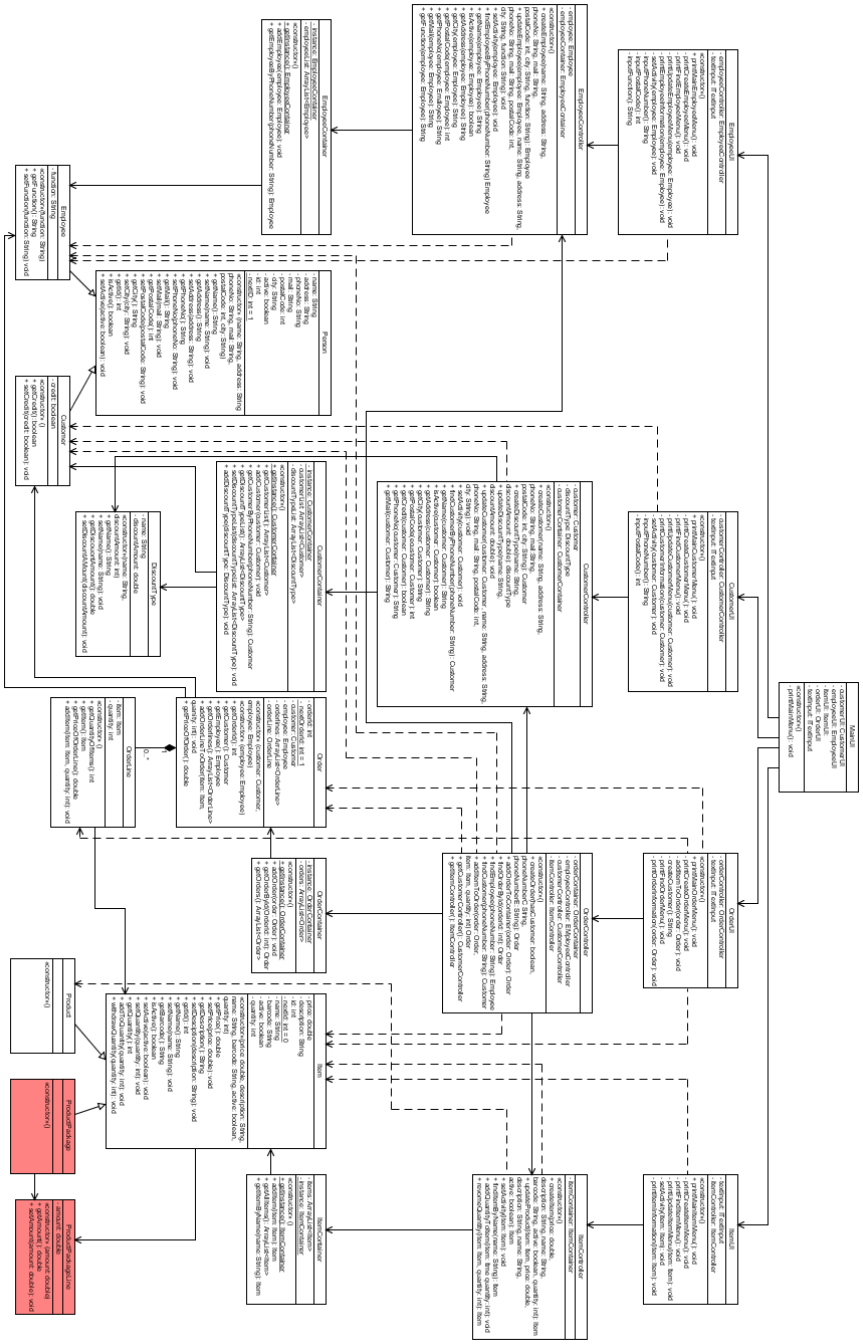


Figure A.1: Design klasse diagram

A.3 Vejledning til system

Guide til at bruge programmet. Før der kan laves en ordre, skal der oprettes en medarbejder, her kan man følge instruktionerne som programmet giver.

- Husk telefonnummeret, da det skal bruges senere.

Til en ordre skal der også være nogle produkter i systemet, de bliver lavet igennem Item Menuen hvor man også kan følge instruktionerne på skærmen.

- Husk navnet på produktet, da det skal bruges senere.
- Der kan laves alle de produkter som man synes er nødvendige.

Nu kan der oprettes en ordre igennem ordre menuen, hvis man gerne vil have en kunde på ordren kan den oprettes igennem Main Menuen, det er dog også muligt at gøre det når ordren oprettes, der skal et telefonnummer som ikke er i systemet indtastes. Igen kan man følge instruktionerne som programmet giver.

- Når man ikke vil tilføje flere produkter kan man "Continue" og ordren bliver printet og gemt
- Hvis man vil se en ordre man har lavet skal man skrive dens ID, id'et bliver generet fra 1, dvs at den første ordre har id'et 1 og stiger med 1 for hvert ordre der bliver lavet.