

# SmartOrder

*Datamatiker*

*University College Nordjylland*

*6. juni 2017*



Casper Froberg Andersen  
Tobias Andersen  
Stefan Krabbe Johansen  
Mikkel Lindstrøm Paulsen  
Arne George Ralston





**University College Nordjylland**

Sofiendalsvej 60

9200 Aalborg SV

UCN

<http://www.ucn.dk/Default.aspx>

**Titel:**

SmartOrder

**Tema:**

Ordrestyring

**Projekt periode:**

1. februar - 6. juni 2017

**Projekt gruppe:**

dmab0916 - gruppe 1

**Deltagere:**

Casper Froberg Andersen

Tobias Andersen

Stefan Krabbe Johansen

Mikkel Lindstrøm Paulsen

Arne George Ralston

**Vejleder:**

Gunhild Marie Andersen

Lise Klitsgaard

István Knoll

**Antal anslag:**

88.964/96.000 tegn med mellemrum

**Antal sider:**

145

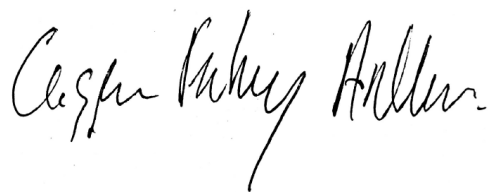
**Abstract:**

This report is written by group 1 from the class dmab0916 at University College of Northern Denmark (UCN). The report is made in close cooperation with Brovst Bolighus.

The content of the report contains analysis, system development and programming principles taught in the first and second semester of the AP graduate education of UCN. The report contains an analysis of the company and documentation for the design and programming of the system. The program is written in Java and SQL.

*Rapportens indhold er frit tilgængeligt, men offentliggørelse (med kildeangivelse) må kun ske efter aftale med forfatterne.*





---

Casper Froberg Andersen  
1061832@ucn.dk



---

Tobias Andersen  
1061912@ucn.dk




---

Stefan Krabbe Johansen  
1062358@ucn.dk



---

Mikkel Lindstrøm Paulsen  
1061928@ucn.dk



---

Arne George Ralston  
1061679@ucn.dk



# Forord

Denne rapport er udarbejdet af gruppe 1 fra dmab0916 ved University College Nordjylland (UCN) på andet semester. Formålet med rapporten er at dokumentere den proces som gruppen har gennemgået i forhold til at implementere et nyt IT-system hos en virksomhed.

Til versionsstyring af programmet er der brugt SVN. Linket til denne er [https://kraka.ucn.dk/svn/dmab0916\\_2Sem\\_6/](https://kraka.ucn.dk/svn/dmab0916_2Sem_6/), og det endelige system er på revisionsnummer 493. En vejledning til at benytte systemet er vedlagt som bilag I.

## Læsevejledning

Der vil gennem rapporten fremtræde kildehenvisninger efter Vancouver-metoden, hvilket vil sige at i teksten står der f.eks. [1], hvilket refererer til kilde nummer 1 i litteraturlisten. I litteraturlisten er bøger og artikler angivet med forfatter, titel og forlag, medens internetsider er angivet med forfatter, titel og dato. Figurer og tabeller er nummereret i henhold til kapitel. Det vil sige i kapitel to har figur nummer et figurnummer 2.1.

I gennem rapporten bruges udtrykkene *vare* og *produkt* som synonymmer. I afsnittet Forundersøgelse bruges *vare*, men i de resterende afsnit benyttes hovedsageligt *produkt*. Dette skyldes brugen af engelske termer i designprocessen, hvor *product* og *produkt* er let sammenlignelige.

*Produkttype* henviser til hvilken type en vare er. En produkttype har en type-attribut som i rapporten refereres til som *grundtype*. Denne attribut beskriver om den givne produkttype er af typen: *hyldevare*, *specialdesign* eller *modul til specialdesign*.

University College Nordjylland, 6. juni 2017





# Indhold

<b>1</b>	<b>Indledning</b>	<b>3</b>
1.1	Problemformulering . . . . .	3
1.2	Succeskriterier . . . . .	4
<b>2</b>	<b>Forundersøgelse</b>	<b>5</b>
2.1	Nuværende situation . . . . .	5
2.2	Strategianalyse . . . . .	11
2.3	IT-strategi . . . . .	12
2.4	Applikationer og information . . . . .	15
2.5	IT-handlingsplan . . . . .	16
2.6	Cost-benefit analyse . . . . .	16
2.7	Brugerdeltagelse . . . . .	21
2.8	Konklusion på forundersøgelsen . . . . .	21
<b>3</b>	<b>Inception</b>	<b>23</b>
3.1	Interview . . . . .	23
3.2	Mock-ups . . . . .	24
3.3	Medarbejder - opgave - mål tabel . . . . .	26
3.4	Use-case diagrammer . . . . .	30

3.5	Use-case beskrivelser . . . . .	32
3.6	Proof of concept . . . . .	33
3.7	Use-case prioritering . . . . .	35
3.8	Use-case fully dressed . . . . .	36
3.9	Domænemodel . . . . .	39
3.10	Konklusion på inceptionfasen . . . . .	41
<b>4</b>	<b>Elaboration</b>	<b>43</b>
4.1	Iteration 1 - Opret salgsordre . . . . .	43
4.2	Iteration 2 - Opret produkt . . . . .	54
4.3	Iteration 3 - Opret produkttype . . . . .	59
4.4	Kode . . . . .	66
4.5	Test . . . . .	70
4.6	Usability-evaluering . . . . .	78
<b>5</b>	<b>Diskussion</b>	<b>81</b>
5.1	Virksomhed . . . . .	81
5.2	Systemet . . . . .	81
5.3	Proces . . . . .	84
5.4	Hvad vi har lært . . . . .	88
<b>6</b>	<b>Konklusion</b>	<b>91</b>
	<b>Litteratur</b>	<b>93</b>
<b>A</b>	<b>Metode</b>	<b>95</b>
A.1	Kotter . . . . .	95
A.2	McGregor's X- og Y-teori . . . . .	97

A.3	Adizes lederroller . . . . .	97
A.4	SWOT-analyse . . . . .	98
A.5	Ansoffs vækstmatrix . . . . .	99
A.6	Information Economics . . . . .	100
A.7	Forandringsstrategier . . . . .	102
A.8	Medarbejder - opgave - mål tabel . . . . .	103
A.9	Workflow . . . . .	103
A.10	Use-case . . . . .	104
A.11	Domænemodel . . . . .	105
A.12	System kontrakter . . . . .	106
<b>B</b>	<b>Cost-benefit beregninger</b>	<b>109</b>
<b>C</b>	<b>Parker Benson</b>	<b>113</b>
<b>D</b>	<b>Brugervenlighed</b>	<b>115</b>
<b>E</b>	<b>Usability test</b>	<b>119</b>
<b>F</b>	<b>Forandringsstrategi</b>	<b>127</b>
<b>G</b>	<b>Brugerevaluering af SmartOrder</b>	<b>131</b>
<b>H</b>	<b>Systemtests</b>	<b>133</b>
<b>I</b>	<b>Brugervejledning</b>	<b>137</b>
<b>J</b>	<b>Create scripts til databasen</b>	<b>141</b>



# 1 | Indledning

Dette projekt tager udgangspunkt i et samarbejde med en virksomhed om at designe og udvikle et IT-system. Arbejdsprocessen tager afsæt i Unified Process (UP), hvilket gør, at rapporten er opbygget om de forskellige iterationer, som projektet har gennemløbet.

Årsagen til, at der skal udvikles et nyt IT-system i samarbejde med en virksomhed, er, at mange virksomheder i dag har udfordringer der kan løses ved hjælp af IT. Eksempelvis er der mange mindre virksomheder der enten ikke har noget IT-system eller at deres nuværende mangler funktionalitet.

Til dette projekt er der samarbejdet med Brovst Bolighus, der til dagligt handler med møbler og andet boliginventar. På skrivende tidspunkt har virksomheden tre problemstillinger, som muligvis kan løses ved hjælp af IT-systemer. Den første problemstilling er, at det tager lang tid at bestille varer fordi de har mange varer hvor der skal bruges specielle varenumre, når varen skal bestilles hjem. Den anden problemstilling er, at de ikke ved, hvor deres kunder kommer fra, og derfor ved de heller ikke, hvor de skal lave deres markedsføring. Den sidste problemstilling er, at det sommetider kan være svært at finde varer på deres lager, da de også bruger butikken som lager.

## 1.1 Problemformulering

De førnævnte problemstillinger resulterer i følgende problemformulering:

*Hvordan kan der udvikles et IT-system, som gør det nemmere for Brovst Bolighus at håndtere bestilling af specialdesignede møbler, finde ud af hvor der skal markedsføres og lettere finde varer på lageret?*

## 1.2 Succeskriterier

For at sikre at problemformuleringen løses på en tilfredsstillende måde, opstilles der en række succeskriterier, lavet af personerne bag projektet, som skal bruges til at vurdere, hvorvidt systemet vil være brugbart for Brovst Bolighus:

1. Systemet er stabilt og indeholder ingen kritiske fejl eller pludselige nedbrud.
2. Systemet hjælper de ansatte med bestilling af varer.
3. Systemet skal målbart nedsætte tiden det tager at udføre opgaver der tidligere blev udført på papir.
4. Systemet skal kunne fortælle, hvor en markedsføringskampagne rammer flest mulige kunder fra målgruppen.
5. Systemet skal til enhver tid kunne fortælle en medarbejder, om en vare er på lager eller hvornår den forventes at ankomme.

For at finde frem til hvilket af de tre del-systemer der er mest kritisk for virksomheden analyseres Brovst Bolighus i næste kapitel med fokus på, virksomhedens nuværende situation og hvor arbejdsopgaverne kan optimeres ved hjælp af et IT-system.

## 2 | Forundersøgelse

I dette kapitel analyseres Brovst Bolighus' nuværende situation. Derudover vil der analyseres, hvordan Brovst Bolighus kan øge sin vækst og effektiviteten af arbejdsopgaver ved hjælp af en strategianalyse. Ydermere vil der fremlægges en IT-strategi, som kan passe til Brovst Bolighus. Desuden findes der frem til, hvilket delsystem der er mest kritisk for Brovst Bolighus.

### 2.1 Nuværende situation

I dette afsnit beskrives Brovst Bolighus og de udfordringer, som virksomheden står med i dagligdagen. Dette gøres ved at opstille virksomhedens problemstillinger, ledernes karakteristika, organisationsstruktur og virksomhedens kultur.

#### 2.1.1 Virksomhedskarakteristik

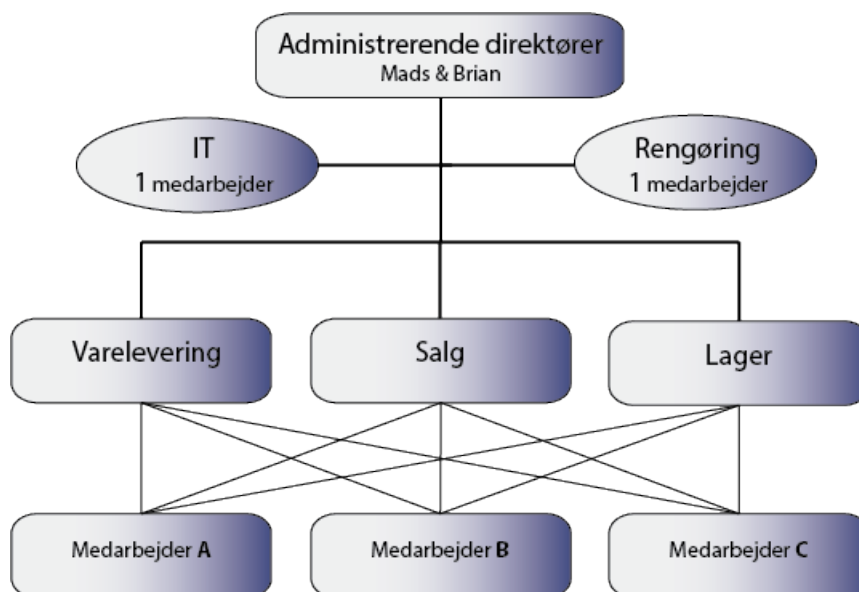
Brovst Bolighus er en anpartsvirksomhed som blev startet af Flemming Johansen i 1994. De nuværende direktører er Brian Krabbe Johansen og Mads Fisker Thorsen, som begge har været med fra starten. Brian indtrådte i ledelsen i år 2000 og Mads i 2003.

I dag har Brovst Bolighus i alt syv ansatte og et salgsareal på cirka 2900 kvadratmeter. Sortimentet består hovedsageligt af møbler, hvoraf flere kan tilpasses efter den individuelle kundes ønsker. Derudover sælges der varer som vægkunst, plaider, puder og brugskunst, der kan bruges til at indrette hjemmet.

### 2.1.2 Organisationsstruktur

I dette underafsnit udarbejdes en organisationsplan, for at give et overblik over, hvordan Brovst Bolighus' organisationsmæssigt er opbygget.

Som det fremgår af organisationsplanen på figur 2.1, er der i alt syv personer, som arbejder i Brovst Bolighus. Ledelsen består af de to direktører, Mads og Brian, som også står for den daglige drift i de forskellige afdelinger og stabe.



Figur 2.1: Organisationsplan over Brovst Bolighus.

Organisationsplanen viser den vertikale og horisontale arbejdsdeling. Den vertikale arbejdsdeling viser, hvordan ledelsesansvaret er fordelt i organisationen. Den horisontale arbejdsdeling viser, hvordan arbejdet er fordelt mellem medarbejderne [1, s. 274-282].

Brovst Bolighus benytter sig af både det funktionelle, funktions og linje-stabs princippet. Dette kommer til udtryk ved, at der er flere medarbejdere som arbejder på tværs af arbejdsopgaver, men også at medarbejderne er samlet i afdelinger. Linje-stabsprincippet ses ved, at der er to stabe og tre afdelinger som ledes af de administrerende direktører. Ydermere viser figur 2.1, at arbejdsopgaverne i Brovst Bolighus er opdelt i tre afdelinger som udfører hver deres funktion. Dette indikerer at virksomhedens horisontale arbejdsdeling anvender funktionsprincippet.



### 2.1.3 Vurdering af organisationsstruktur

Efter samtale med Mads er Brovst Bolighus som virksomhed blevet vurderet til at være organisk, hvilket vil sige, at den enkelte medarbejder ikke har nogle fast definerede arbejdsopgaver, men derimod en generel ansvarlighed (jvf. det funktionelle princip beskrevet i afsnit 2.1.2). Derudover er organisationen styret gennem information og rådgivning, ikke instrukser fra overordnede hvilket gør, at den enkelte medarbejder får et større engagement i organisationens opgaver, og bidrager til vækst og effektivitet [1, s. 254].

### 2.1.4 Organisationskultur

For at finde ud af hvordan ledelsen og medarbejderne omgås hinanden, analyseres i det følgende afsnit lederne af Brovst Bolighus. Dette gøres for at give et overblik over, hvordan arbejdskulturen er i virksomheden.

#### Lederkarakteristik og menneskesyn

For at beskrive begge ledere bliver de først kategoriseret efter McGregors X- og Y-teori, som antager, at der findes to forskellige menneskesyn for ledere. Metoden er nærmere beskrevet i bilag A.2.

Både Brian og Mads anses for at have et Y-syn som ledere. Dette skyldes, at de begge synes, at yde en indsats er naturligt, godt kan lide at give medarbejderen frihedsgrader og har en forventning om, at medarbejderne er villige til at påtage sig et ansvar. Derudover giver de lov til, at medarbejderen kan få dækket deres ego- og selvrealiseringsbehov ved hjælp af jobbet fordi de kan styre deres egen hverdag.

#### Adizes lederroller

I dette afsnit beskrives lederne ud fra Adizes' lederroller, som er beskrevet i bilag A.3. De roller, som Brian og Mads hver især er blevet tildelt, fremgår af tabel 2.1.

Adizes 4 lederroller	
Brian	PAeI
Mads	paEI

**Tabel 2.1:** Adizes lederroller for Brovst Bolighus.

Brian er blevet tildelt store bogstaver i producent-, administrator- samt integratorrollen. Dette skyldes, at han er meget resultat- og handlingsorienteret, har et stort præstationsbehov, altid er flittig og travl og har en stor faglig viden. Derudover har Brian en systematisk og organiseret måde at arbejde på. Desuden skaber han motivation og korpsånd og udvikling hos medarbejderne.

Mads er blevet tildelt store bogstaver i entrepreneur- og integratorrollen, da han hele tiden forsøger at forbedre det eksisterende, finde nye produkter og er risikovillig. Ydermere leder han gennem teamwork og skaber motivation, korpsånd og udvikling hos medarbejderne.

Ved hjælp af disse vurderinger konkluderes det, at Brovst Bolighus har udfyldt alle fire lederroller, hvilket gør, at der er et godt lederskab i virksomheden.

### 2.1.5 Problemer, hypoteser og løsningsmuligheder

Brovst Bolighus' datalogiske udfordringer er opstillet på tabel 2.2, hvorefter hypoteser på problemstillingerne opstilles, samt en løsningsmulighed til disse foreslås.

Problemer	Hypoteser	Løsningsmuligheder
Mange sammensætningsmuligheder ved visse møbler	Medarbejderne har svært ved at beskrive sammensætningen af kombinationsmuligheder	IT-system der overskueliggør kombinationsmuligheder
Der er ikke 100 % styr på lageret	Medarbejderne har ikke mulighed for at vide, hvad der præcist er på lager	IT-system der overskueliggør lagerbeholdning
Vil gerne vide hvor deres kunder kommer fra	Ledelsen har ikke styr på, hvor de forskellige kunder kommer fra	IT-system med kundekartotek
Har ikke overblik over kundegrupper	Ledelsen ved ikke, hvilke kundegrupper, der handler hos virksomheden mest	IT-system til at holde styr på kundegrupper
Virksomheden ved ikke, hvor og hvordan de skal reklamere	Ledelsen ved ikke, hvor eller hvor godt deres reklamer virker	IT-system til at registrere hvor kunder kommer fra
Bestillinger af speciallavede sofaer er en udfordring, da der ikke bruges varenumre men beskrivelser	Det er uoverskueligt for medarbejderen at bestille en sofa ved leverandøren, da beskrivelser er sværere at forstå end varenumre	IT-system til varebestilling

**Tabel 2.2:** Tabel over identificerede problemer, hypoteser og løsningsmuligheder

### 2.1.6 SWOT-analyse

Ved nærmere kendskab til Brovst Bolighus som virksomhed er der forskellige faktorer, som kan blive belyst vha. en SWOT-analyse. Af selvsamme årsag udarbejdes en SWOT-analyse i dette afsnit. Metoden, som benyttes, er nærmere beskrevet i bilag A.4.

Interne forhold	
Stærke sider (Strengths)	Svage sider (Weaknesses)
<ul style="list-style-type: none"> <li>• Solid egenkapital</li> <li>• 23 års erfaring</li> <li>• Delt ansvar mellem medarbejdere</li> <li>• 2.900 kvadratmeter butik</li> <li>• Stort udvalg af special møbler</li> <li>• Har selv midler til yderligere ekspansion</li> <li>• Levering af alle slags varer</li> <li>• Medarbejdere fra lokalsamfundet</li> <li>• Stor lokal opbakning</li> <li>• Stor viden om produkter</li> <li>• En del af Danbo-kæden</li> </ul>	<ul style="list-style-type: none"> <li>• Lageroverblik</li> <li>• Ingen idé om hvor kunderne har hørt om virksomheden</li> <li>• Butikken er i et villakvarter, hvor det godt kan være lidt svært at finde den</li> <li>• Begrænset brug og kendskab til IT</li> <li>• Bestilling af varer hos leverandør</li> <li>• Har ikke et samlet IT-system</li> </ul>
Eksterne forhold	
Muligheder (Opportunities)	Trusler (Threats)
<ul style="list-style-type: none"> <li>• Nye produkter</li> <li>• Nemmere bestilling ved leverandør</li> <li>• Mere net handel</li> <li>• Loyale kunder</li> <li>• Mulighed for ekspansion</li> <li>• Nye og flere reklamemuligheder</li> </ul>	<ul style="list-style-type: none"> <li>• Konkurrenter</li> <li>• Recession</li> <li>• Stigende reklameomkostninger</li> <li>• Stigende transportomkostninger</li> <li>• Stigende priser hos leverandører</li> </ul>

Tabel 2.3: SWOT-analyse for Brovst Bolighus.

SWOT-analysen, tabel 2.3, ses der blandt virksomhedens svage sider, at varebestilling er tidskrævende og uoverskuelig, samt de ikke har noget IT-system til at understøtte dette. Disse to elementer kan siges at være tæt forbundne. Ydermere foreligger der oplagte muligheder for virksomheden i form af procesoptimering, f.eks. ved implementering af IT-system til at varetage komplekse leverandør-bestillinger knyttet til salg af specialdesignede møbler.

Da specialdesignede varer er en kompleks og tidskrævende proces, tilskyndes at Brovst Bolighus intensiverer udviklingen af de interne forhold med særlig fokus på procesoptimering. Dette viser, at Brovst Bolighus bruger den moderne designstrategi [1, s. 350].

Det tilrådes derfor virksomheden at komplimentere vækststrategien med en mini-maxi strategi til at minimere svaghederne og maksimere mulighederne, hvor fokus vil være på at optimere processer der vil hjælpe virksomheden med at nå sine mål.

I næste afsnit vil der analyseres, hvorledes Brovst Bolighus kan øge markedsandele på sine nuværende markeder og lave markedsudvikling.

## 2.2 Strategianalyse

I dette afsnit analyseres, hvilke markeder Brovst Bolighus har indflydelse på, og hvordan Brovst Bolighus kan penetre nye markeder. Ydermere vælges en konkurrencestrategi for at se, hvor Brovst Bolighus kan få størst vækst.

### 2.2.1 Ansoffs vækstmatrice

I dette afsnit analyseres Brovst Bolighus' muligheder for at vækste ved hjælp af Ansoffs vækstmatrice, som står beskrevet i bilag A.5. Dette gøres for at få en forståelse for, hvordan Brovst Bolighus kan øge sin vækst.

		Produkter	
		Nuværende	Nye
Markeder	Nuværende	Markedspenetrering - Specialdesignede møbler - Masseproducerede varer	Produktudvikling - Designer møbler - Discount møbler
	Nye	Markedsudvikling - Udvidet aldersmålgruppe - Udvidet geografisk område	Diversifikation - Isenkram - Udlejning af møbler - Indretningshjælp

Figur 2.2: Ansoffs vækstmatrice

På figur 2.2 ses vækstmatricen for Brovst Bolighus. Det fremgår, at virksomhedens markedspenetreringsstrategi består af deres specielt designede møbler og masseproducerede varer. En anden mulighed for vækst er at udvide målgruppen med flere aldersgrupper. Dette kan blandt andet gøres ved at udvide deres møbelsortiment til at indeholde flere billige møbler, da det ville kunne tiltrække en yngre kundesgruppe, som eksempelvis kan være på udkig efter møbler til deres første lejlighed.

### 2.2.2 Valg af strategi

Efter de foregående afsnits analyser og interview med virksomheden vurderes det, at Brovst Bolighus adskiller sig fra konkurrenterne ved at have mange møbler, som kan tilpasses den enkelte kundes ønsker sideløbende med virksomhedens udbud

af standard møbler i forskellige prisklasser. Desuden lægger Brovst Bolighus stor vægt på at servicere sine kunder. Der er således tale om en konkurrencestrategi, hvor der i stigende grad satses på differentiering, dog med bibeholdelse af evnen til at betjene en relativ stor målgruppe med varer til konkurrencedygtige priser. For at understøtte dette anbefales det, at Brovst Bolighus' umiddelbare vækststrategi skal være at satse på markedspenetrering, da der foreligger mulighed for at fortrænge konkurrenterne med specialdesignede varer og deres meget kundeorienterede fokus.

### 2.2.3 Idé, mission, vision og målsætning

Idéen bag Brovst Bolighus er at kunne tilbyde kvalitetsmøbler til den enkelte kunde, som de fleste kunder har råd til.

Brovst Bolighus' mission lyder *"Det er Danbo Møblers mission at tilbyde vores kunder markedets bedste priser på møbler - du er derfor sikret en god handel hos os."* [2].

Virksomhedens vision er at tjene penge ved at sælge de møbler der kan sælges til den enkelte kunde med størst muligt profit.

Deres målsætning er at blive den førende møbelvirksomhed ved at udvide sin kundekreds og sit varesortiment.

### 2.2.4 Delkonklusion

Strategien for Brovst Bolighus bør tilrettelægges så den nuværende markedspenetrering bibeholdes, samtidigt bør der fokuseres på yngre individer, og i den forbindelse bør der overvejes produkter som er både masseproduceret og billige. Dette bør være produkter som er hurtigt omsættelige. Dette vil give større markedsandele, samt andel på et marked hvor virksomheden på nuværende tidspunkt ikke fokuserer.

## 2.3 IT-strategi

I dette afsnit vil der fremlægges en hensigtsmæssig IT-strategi som understøtter Brovst Bolighus forretningsstrategi og vision. Dette gøres ved først at se på virksomhedens nuværende IT-situation. Dernæst vil der på baggrund af tidligere afsnits analyser undersøges, hvilke IT-muligheder der foreligger.

### 2.3.1 Nuværende IT-system

På nuværende tidspunkt benytter Brovst Bolighus sig af et almindeligt mail system til at kommunikere med deres leverandører. Varerne, der bestilles hjem, bliver enten placeret i butikken, på lageret, eller kørt ud til kunden ved først givne lejlighed. Det bliver ikke registreret i noget IT-system, at en vare har været på matriklen.

Desuden benyttes der et system til at gemme fakturaer og kundeoplysninger, men disse oplysninger bliver ikke benyttet til andet, end at kunne genfinde en faktura.

### 2.3.2 Fastlæggelse af IT-muligheder

Der er hos Brovst Bolighus en udfordring i forhold til det fysiske lager. Ca. 70 % af lageret er fremvist som udstilling i den næsten 2900 m<sup>2</sup> store butik. De sidste 30 % står på et lager, hvor der ikke er adgang for kunder. Der er dog ikke et system til at holde styr på, hvad der præcist er på lager, og hvad der er i udstillingen. Dette betyder, at lederne typiske er nødt til at have et overblik over, hvad der er på lager. Dette gør, at de aldrig er 100 % sikre på hvad der præcis er i butikken, medmindre de går rundt og tjekker. Dette kan løses med et IT-lagersystem.

Brovst Bolighus ved hvor deres kunder kommer fra, såfremt de køber noget, da de registrerer købernes adresse. Brovst Bolighus ønsker at benytte dette data til at finde ud af, hvor deres største kundegrupper befinder sig. Ydermere har Brovst Bolighus heller ikke nogle deciderede kundegrupper, hvilket betyder, at de ikke har en måde, hvorpå de kan opdele kunderne i grupper. Brovst Bolighus ønsker at have en måde at inddele kunder i grupper, som gør det muligt at udspecificere hvad de enkelte grupper køber. Dette kan løses med et IT-system som kan inddele kunder efter forskellige indstillinger, præferencer eller køb.

En af de store udfordringer, som Brovst Bolighus har, er, at de ikke ved, hvor og hvordan de skal reklamere. Dette relaterer til det forrige problem, med de ikke ved hvordan de besøgende kunder kender til Brovst Bolighus. Udfordringen er derfor at udvikle et system eller et helt nyt koncept, som kan hjælpe Brovst Bolighus med at finde ud af, hvor deres besøgende har fundet information om virksomheden.

Et andet problem er, at ordrehåndtering af speciallavede sofaer er en stor udfordring. Dette skyldes, at de ikke bruger alle varenumrene, men blot beskriver dele af varen med varenumre. Dette skyldes at det tager for lang tid at finde alle de korrekte varenumre. Dette er ikke blot en udfordring for Brovst Bolighus i forbindelse med en salgsordre, men også for leverandøren i forbindelse med bestillingsordre,

da de ved at bruge alle de korrekte varenumre ved præcis hvilke varenumre der skal benyttes til at lave den nye sofa.

### 2.3.3 Nøgleapplikationer

Ud fra beskrivelsen af IT-muligheder i afsnit 2.3.2 forekommer det, at Brovst Bolighus har flere problemer, som kan løses ved hjælp af IT. På baggrund af de foregående undersøgelser kan disse problemstillinger løses ved hjælp af IT, som kan indeholde følgende nøgleapplikationer:

- Lagerstyring
- Håndtering af ordrer
- System til markedsføring

### 2.3.4 Implementering

I dette afsnit beskrives der, hvilke hensyn der tages højde for. Der er flere muligheder for implementering, og det er derfor vigtigt at lægge fokus på, at overgangen skal være så glidende som mulig. Overordnet kan det siges, at implementering skal tage hensyn til virksomhedens struktur, teknologi, opgaver og personale.

Med struktur menes der, at implementering af det nye system ikke må komme i konflikt med virksomhedens arbejdsdeling, kommunikation og beslutningsstruktur. Da IT-systemet skal benyttes på alle ansættelsesniveauer i virksomheden, skal der ikke implementeres nogle adgangsrestriktioner. Dog kan det tænkes at der skal tilføjes nogle forretningsregler som kun må redigeres af ledelsen.

Den nye teknologi må ikke komme i konflikt med den eksisterende teknologi. Med dette menes der redskaber, der bruges til at løse arbejdsopgaver som f.eks. systemer, maskiner og arbejdsprocesser. Da Brovst Bolighus ikke har noget nuværende IT-system, som det nye system skal integreres med, er der ikke nogle umiddelbare teknologiske hensyn at tage ved dette punkt.

Da virksomheden benytter sig af det funktionelle organisations princip jvf. afsnit 2.1.2, vil det nye system blive benyttet af flertallet af de ansatte. Derfor vil det være hensigtsmæssigt at give alle medarbejdere grundig træning i at bruge det nye program som en del af implementeringen.



### **Implementeringsmuligheder**

Implementering kan være en total udskiftning over en weekend eller helligdag, hvor butikken er lukket, da det giver ro til opstilling af back-up systemer, computere der skal anvendes af medarbejdere og ledere samt installation af soft- og hardware.

Alternativt kan det implementeres sideløbende, hvor det nuværende system stadig er i brug, mens det nye bliver installeret. Dette betyder, at implementeringen foregår i virksomhedens almindelige åbningstid, og at der ikke skal kaldes en medarbejder ind uden for den almindelige åbningstid.

## **2.4 Applikationer og information**

I dette afsnit beskrives der, hvad der er nødvendigt i forhold til at implementere et nyt system hos Brovst Bolighus.

### **2.4.1 På kort sigt**

For at få systemet op at køre er det nødvendigt at have en eller flere computere, hvilket de allerede har. Investering er derfor ikke nødvendig.

Dog skal der købes andet hardware i form af en server, der kan holde styr på de forskellige varenumre der findes til de specialdesignede møbler. For at muliggøre dette er det nødvendigt at have plads til alt data, som eksisterer i øjeblikket.

Desuden er det nødvendigt at have en back-up af dataene, sådan alt data ikke går tabt, hvis der skulle ske noget uventet. Dette kan til at starte med være en "on-site" løsning.

### **2.4.2 På længere sigt**

Når der kommer nye og flere leverandører, er det nødvendigt at udvide serverkapaciteten og dermed størrelsen af backup-mediet.

Det er muligt at benytte en anden lokalitet til back-up enheden for at sikre data mod lokale katastrofer.

## 2.5 IT-handlingsplan

I dette afsnit opstilles de applikationer, som blev beskrevet i afsnit 2.3.3, der vurderes vigtigst for Brovst Bolighus daglige arbejdsgange.

### 2.5.1 Rækkefølge af applikationer

I dette afsnit beskrives den logiske rækkefølge af de forskellige nøgleapplikationer, som er beskrevet i afsnit 2.3.3.

Det logiske vil være først at udvikle markedsførings-delsystemet, dernæst ordrehåndterings-delsystemet og til sidst lager-delsystemet.

Markedsførings-delsystemet skal udvikles først, fordi det vil give mulighed for at trække flere kunder til virksomheden. Årsagen til at dette delsystem skal udvikles er, at det er flere og flere som siger "Nej tak" til reklamer [3] og det derfor er sværere at finde ud af hvor Brovst Bolighus bør markedsføre sig.

For at akkommodere de nye kunder, som kommer ved hjælp af markedsførings-delsystemet, er det oplagt at lave et ordrehåndterings-delsystem til at håndtere alle de nye bestillinger og ordrer.

Til sidst for at holde styr på alle de forskellige varer, giver det god mening for Brovst Bolighus at få udviklet et lager-delsystem. Det er dog vigtigt at huske, at 70 % af virksomhedens lager står i butikken, og Brovst Bolighus derfor har en lav lagerservicegrad [4, s. 5].

## 2.6 Cost-benefit analyse

I dette afsnit laves der en cost-benefit (CB) analyse på de tre mulige delsystemer. Analysen bruges til at beslutte, hvilket af delsystemerne der er mest kritisk for virksomheden og dermed er vigtigst at få udviklet. En CB analyse laves ud fra estimer om, hvad der forventes at kunne ske af forbedringer for virksomheden ved implementering af nyt IT-system. Nogle af estimerne har flere beregninger bag sig, som er vedlagt i bilag B.

Alle priser er inkl. moms	Ordrehåndterings-delsystem		
År	1	2	3
<b>Direkte målelige omkostninger</b>			
Udviklingsomkostninger (engangsydelse)	487.560,00	0,00	0,00
Server med harddisk (engangsydelse)	2605,90	0,00	0,00
<b>Driftsomkostninger</b>			
Strøm til server (kr. pr. år)	3.009,06	3.009,06	3.009,06
Support og vedligeholdelse (kr. pr. år)	27.000,00	27.000,00	27.000,00
<b>Samlede udgifter (kroner)</b>	520.174,96	30.009,06	30.009,06
<b>Direkte besparelser</b>			
Salg til flere kunder (bruttofortjeneste, målt i kr. pr. år)	282.900,00	282.900,00	282.900,00
<b>Samlede indtægter/besparelser (kroner)</b>	282.900,00	282.900,00	282.900,00
<b>Difference (kroner)</b>	-237.274,96	252.890,94	252.890,94
<b>Tilbagebetalt (kroner)</b>	-237.274,96	15.615,98	268.506,92

**Tabel 2.4:** Return of investment for ordrehåndterings-delsystemet.

Tabel 2.4 viser den forventede gevinst ved at investere i ordrehåndterings-delsystemet. I dette delsystem vil der være overskud allerede fra andet år, og der vil være 'break even' inden der er gået to år. Grunden til, at der kan komme mere salg til kunder, er at der er mere tid til at flere kunder kan få en god service og der ikke skal bruges så meget tid til at oprette nye og holde styr på eksisterende salgsordrer.

Alle priser er inkl. moms	Markedsførings-delsystem		
År	1	2	3
<b>Direkte målelige omkostninger</b>			
Udviklingsomkostninger (engangsydelse)	195.042,00	0,00	0,00
Server med harddisk (engangsydelse)	2605,90	0,00	0,00
<b>Driftsomkostninger</b>			
Strøm til server (kr. pr. år)	3.009,06	3.009,06	3.009,06
Support og vedligeholdelse (kr. pr. år)	16.200,00	16.200,00	16.200,00
<b>Samlede udgifter (kroner)</b>	216.838,96	19.209,06	19.209,06
<b>Direkte besparelser</b>			
Salg til flere kunder (bruttofortjeneste, målt i kr. pr. år)	100.000,00	848.700,00	848.700,00
<b>Samlede indtægter/besparelser (kroner)</b>	100.000,00	848.700,00	848.700,00
<b>Difference (kroner)</b>	-116.838,96	829.490,94	829.490,94
<b>Tilbagebetalt (kroner)</b>	-116.838,96	712.651,98	1.542.142,92

**Tabel 2.5:** Return of investment for markedsførings-delsystemet.

Tilbagebetalingsberegningen for markedsførings-delsystemet er vist på tabel 2.5. Salg til flere kunder første år er "kun" på 100.000 kroner skyldes, at der skal indsamles data om kunderne, før markedsføringen kan blive mere målrettet. Efter det første år med indsamlet data kan salget stige meget, og der forventes også at 'break even' allerede inden der er gået 2 år, da Brovst Bolighus så ved, hvor de skal investere i deres markedsføring.

Alle priser er inkl. moms	Lager-delsystem		
År	1	2	3
<b>Direkte målelige omkostninger</b>			
Udviklingsomkostninger (engangsydelse)	65.008,00	0,00	0,00
Server med harddisk (engangsydelse)	2605,90	0,00	0,00
<b>Driftsomkostninger</b>			
Strøm til server (kr. pr. år)	3.009,06	3.009,06	3.009,06
Support og vedligeholdelse (kr. pr. år)	5.400,00	5.400,00	5.400,00
<b>Samlede udgifter (kroner)</b>	76.022,96	8.409,06	8.409,06
<b>Direkte besparelser</b>			
Sparet løn ved at kigge på lager (kr. pr. år)	33.480,00	33.480,00	33.480,00
<b>Samlede indtægter/besparelser (kroner)</b>	33.480,00	33.480,00	33.480,00
<b>Difference (kroner)</b>	-42.542,96	25.070,94	25.070,94
<b>Tilbagebetalt (kroner)</b>	-42.542,96	-17.472,02	7.598,92

Tabel 2.6: Return of investment for lager-delsystemet.

På tabel 2.6 kan den potentielle gevinst for lager-delsystemet aflæses. Dette delsystem vil først give overskud fra tredje år og nå punktet 'break even' allerede inden 2 år. Dette skyldes, at delsystemet er forholdsvis dyrt at udvikle kontra den umiddelbare besparelse. Besparelsen vil dog ses efter tre år, da Brovst Bolighus vil spare tid ved ikke at skulle gå ud på lageret for at tjekke, om varen er på lager.

Som det fremgår af tabellerne, er der stor forskel på de forventede udviklingsomkostninger, indtægter/besparelser og de forventede tilbage-betalingstider for de forskellige delsystemer.

Udover de økonomiske fordele ved at udvikle et nyt system, kan der være mindre indirekte målelige værdier, som eksempelvis er øget arbejdsglæde og mindre pres på den enkelte medarbejder. Ydermere er der mulighed for at spare, da en bestilling hos en leverandør ofte leveres fragtfrit til kunden, hvis bestillingen er stor nok. Det er en problemstilling som ordrehåndterings-delen kan håndtere, ved først at sende bestillingen, når ordren til den pågældende leverandør er stor nok.

For at finde ud af hvilket delsystem der er vigtigst for Brovst Bolighus, vil der i næste afsnit laves en prioritering ved hjælp af Parker Benson-modellen.

### 2.6.1 Parker Benson

I dette afsnit vil der blive opstillet en Parker Benson analyse. Metoden, som benyttes, er beskrevet i bilag A.6, og forkortelserne, der bruges i det følgende, er udspecificeret i tabel 2.7.

Return of Investment (ROI)	Strategic Match (SM)
Competitive Advantage (CA)	Management Information (MI)
Competitive Response (CR)	Organisational Risk (OR)
System Architecture (SA)	Definition Uncertainty (DU)
Technological Uncertainty (TU)	Infrastructure Risk (IR)

**Tabel 2.7:** Nøgleord til Parker Benson.

På figur 2.3 fremgår det, at Brovst Bolighus har vægtet forretningsdomænet højere end teknologidomænet. Desuden er ROI, CA og MI vægtet væsentligt højere sammenlignet med CR og OR, hvilket indikerer at Brovst Bolighus prioriterer afkast af både målelige og svært-målelige værdier og at kunne forblive konkurrencedygtige.

	Business domain						Technology domain				Total
	ROI	SM	CA	MI	CR	OR	SA	DU	TU	IR	
Vægt	5	3	4	4	2	-2	2	-3	-4	-1	
Ordrehåndteringsdelsystem	4	4	3	3	3	5	3	1	2	1	46
Markedsføringsdelsystem	5	4	4	4	4	5	0	4	2	3	44
Lagerstyringsdelsystem	1	1	2	2	1	5	1	1	1	4	7

**Figur 2.3:** Cost-benefit analyse for Brovst Bolighus.

Figuren viser også, at ordrehåndterings- og markedsføringsdelsystemerne vægtes højere end lager-delsystemet.

Ordrehåndteringsdelsystemet vægtes over middel i hele forretningsdomænet, hvilket vil sige, at det vil få en høj score uanset udfaldet i teknologidomænet.

Grunden til markedsføringsdelsystemet vægtes så højt er, at der generelt ikke er nogle store udfordringer og ændringer der skal implementeres, for at Brovst Bolighus kan begynde at bruge det nye system.

Årsagen til, at lager-delsystemet vægtes så lavt, skyldes blandt andet at Brovst Bolighus, som nævnt i afsnit 2.3.2, har 70 % af deres lager i butikken.

OR, beskriver projektets risikograd, i forhold til hvor 'gearet' virksomheden er til at gennemføre projektet. Denne værdi er høj i alle 3 delsystemer, og det skyldes,

at virksomheden ikke er så teknisk anlagt. De er dog omstillingsparate og ønsker et nyt IT-system. Derved betyder det, at virksomheden ikke er bedst muligt 'gearet', dog skyldes dette hovedsageligt, at virksomheden ikke benytter IT-systemer udover et simpelt ordresystem som registrerer en kunde ved salg.

Ud fra denne analyse kan det konkluderes, at selvom markedsføringsdelsystemet har en høj ROI, kan det umiddelbart bedre betale sig at prioritere ordrehåndteringsdelsystemet.

## 2.7 Brugerdeltagelse

I forbindelse med udvikling af et nyt system til Brovst Bolighus er det vigtigt, at det er et brugervenligt system. Derfor er det nødvendigt at en eller flere eksperter gennemfører en analyse af det nye system.

Derfor vil der blive udført en usability-evaluering af systemet i samarbejde med Mads, som gør, at forandringsstrategien er en ekspertstrategi. Metoden, som benyttes, er beskrevet i afsnit A.7. Dette gøres på trods af, at der muligvis kan gå information tabt fra andre ansatte, grundet de ikke er med i udviklingsprocessen.

## 2.8 Konklusion på forundersøgelsen

På baggrund af de problemstillinger, som er blevet opstillet hos Brovst Bolighus i IT-forundersøgelsen, kan det konkluderes, at Brovst Bolighus kan få glæde af at implementere et nyt IT-system. Det fastlægges at de nøgleapplikationer, som er nævnt i 2.3.3, alle anses for at kunne gavne Brovst Bolighus. Dog konkluderes det, at nøgleapplikationen *ordrehåndtering* giver den største forretningsværdi som belyses i afsnit 2.6.1, og af denne årsag lægges fokus på dette delsystem.





## 3 | Inception

I dette kapitel vil der blive analyseret, hvordan det nye system til Brovst Bolig-  
hus skal implementeres. For at designe systemet vil der blive foretaget et interview,  
udviklet mock-ups og analyseret de eksisterende arbejdsgange der er i virksomhe-  
den. Derudover prioriteres der, hvilken use-case der er vigtigst for det nye system  
og der udvikles en domænemodel til det nye system.

### 3.1 Interview

Når det nye ordrehåndterings-delsystem, herfra kendt som SmartOrder, skal ud-  
vikles, er det vigtigt at vide, hvordan brugernes nuværende arbejdsgange er. Dette  
gøres for at kunne optimere arbejdsprocessen, som er forbundet med at bestille  
varer.

Måden, der bestilles varer på, er ved at medejeren, Mads, får et stykke papir med  
en masse modul-koder, fra en medarbejder, som har solgt den pågældende sofa.  
Hver modul-kode fortæller noget om den specialdesignede sofa. Disse koder skal,  
når det er tid til bestilling, indtastes manuelt i en mail, som derefter sendes til  
leverandøren.

En ting, som Mads gerne vil have at det nye program skal kunne er, at det selv  
holder styr på, hvornår det er tid til at sende afgivne ordrer til leverandørerne,  
da ordren leveres fragtfrit, hvis den overstiger et bestemt beløb. Dog skal det også  
være muligt at sende bestillingen selvom der kommer et fragtgebyr.

Ydermere vægter Mads brugervenlighed højt, da der i virksomheden ikke er den  
store tekniske viden.

## 3.2 Mock-ups

I dette afsnit beskrives mock-ups, der er udviklet til implementering af SmartOrder. Formålet med disse mock-ups er at give Brovst Bolighus en idé om, hvordan systemet vil se ud og fungere, når det er implementeret.

Generelt for de udviklede mock-ups er, at de er udviklet efter Gestalt lovene og Jakob Nielsens regler om design af grænseflader, som er beskrevet i bilag D.

The mockup shows a web application for creating orders. The main header is 'Brovst Bolighus ApS' and the page title is 'Opret Ordre'. The interface is divided into several sections:

- Order-info:** Fields for 'Ordrenr:' and 'Dato:'.
- Kunde:** Fields for 'Navn:', 'Adresse:', 'Telefon:', 'By:', 'Post nr.', and 'Email:'. A link 'Opret ny kunde' is below.
- Tilføj vare:** A section with a 'Vare' tab and a 'Specialdesign' sub-tab. It includes a search bar 'Filtrer liste', a 'Varekategori' dropdown, and a table with columns: 'Varenummer', 'Beskrivelse', 'Antal på lager', and 'Pris per styk'. To the right is a product image of a sofa, a 'Beskrivelse:' text area, an 'Antal:' input field, and a 'Tilføj vare til ordre' button.
- Orderlinjer:** A table with columns: 'Varenummer', 'Beskrivelse', 'Antal', 'Pris per styk', and 'Deltotal'.
- Total:** A field for the total amount.
- Buttons:** 'Gem ordre', 'Opret Ordre', and 'Annullere' at the bottom right.

Figur 3.1: Mockup for opret ordre med en almindelig vare.

På figur 3.1 vises en mock-up for opret salgsordre, hvor der skal bestilles en masseproduceret vare. De forskellige informationer er så vidt muligt holdt adskilt, og systemet er forsøgt designet sådan, at brugerne ikke selv skal tænke.

**Brovst Bolighus ApS** Opret Ordre

**Ordre-info**

Ordrenr:

Dato:

**Vare** Tilføj vare

Model:  Beskrivelse:  Kategori:

Leverandør:

Modultype:

Modulnavn	Beskrivelse

Tilføj modul ->

Modulsammensætning:


Tilføj vare til ordre

**Orderlinjer** [Opret ny kunde](#)

Varenummer	Beskrivelse	Antal	Pris per styk	Deltotal

**Total:**

Fjern

Gem ordre
Opret ordre
Annuller

Figur 3.2: Mockup for opret salgsordre med en special-designet vare.

Figur 3.2 viser mock-upen til opret salgsordre, når der skal bestilles en specialdesignet vare.

### 3.2.1 Ekspert review på mock-ups

I forbindelse med at udarbejde de førnævnte mock-ups har der været nogle testpersoner til at lave ekspert review på disse. Testpersonerne havde ikke set brugerfladen før reviewet, hvilket gør det muligt at teste om de forskellige designvalg er så intuitive som tiltænkt. Hver brugertest havde en varighed af 10 minutter, og i alt var der to testpersoner som var i gennem dette review. Ved udførelse af testene blev der valgt en testleder som skulle være ansvarlig for begge tests, samt to observatører som skulle hjælpe med at notere spørgsmål og svar fra testpersonerne. Derudover støttede observatørerne testlederen med uddybende spørgsmål, såfremt det blev nødvendigt. Rapporten, der er blevet udarbejdet fra testen, kan findes i bilag E.

### Delkonklusion

Det kan konkluderes på baggrund af testen, at langt de fleste af de designvalg som blev besluttet tidligt i brugerfladen designfase var gode. Testpersonerne havde nemt ved at navigere i det præsenterede vindue, og beskrivelserne og tabellerne var sigende og godt placerede. Testpersonerne havde forbedringer til designet. Heriblandt ændring af knappers placering for at overholde Windows' standard i UI designet, mere tydelighed om det er et tilbud eller en ordre man er i gang med at lave. Det blev desuden noteret, at testpersonerne ikke kunne finde eller helt overså søgefunktionen.

Ud fra denne test er den færdige brugerfalde blevet forbedret med ovennævnte problematikker fra testpersonerne. Brugerfladen overholder nu Windows' standard for knapplacering, og ordreinformation viser nu, hvad brugeren er i gang med. Søgefunktionen har fået en beskrivende grå tekst i søgefeltet.

### 3.3 Medarbejder - opgave - mål tabel

På nedenstående tabel (tabel 3.1) ses alle opgaver, som en aktør kan udføre. Metoden, som er benyttet til udarbejdelse af tabellen, er beskrevet i bilag A.8.

Aktør	Opgave	Mål	Trin i opgave
Leder	Opdater medarbejder	Medarbejder er blevet opdateret	Find opdater medarbejder i systemet Vælg medarbejder Tilføj ændringer Gem ændringer
Leder	Opret medarbejder	Medarbejder er blevet oprettet	Find opret medarbejder i systemet Tilføj medarbejderoplysninger Færdiggør oprettelsen
Leder	Godkend leverandørbestilling	Bestilling er blevet sendt til leverandør	Find udgående bestillinger i systemet Vælg bestilling(r) Send bestilling(r) til leverandør

Leder	Aflæs kunde-statistik	Kunde-statistikker relevant for markedsføring beregnes og fremvises	Angiv parametre til beregning af statistik Læs statistik Træf markedsføringsbeslutning på baggrund af data
Leder	Aflæs salgs-statistik	Relevante salgsstatistikker beregnes og fremvises	Angiv parametre til beregning af statistik Læs statistik Træf markedsføringsbeslutning på baggrund af data
Leder	Find medarbejder	Medarbejder er blevet fundet	Find søg medarbejder i systemet Søg efter medarbejder Vælg medarbejder
Salgsmedarbejder	Opret kunde	Kunde er blevet oprettet	Find opret kunde i salgsdelsystemet Tilføj kundeoplysninger Færdiggør oprettelsen
Salgsmedarbejder	Opdater kunde	Kunde er blevet opdateret	Find opdater kunde i salgsdelsystemet Vælg kunde Tilføj ændringer Gem ændringer
Salgsmedarbejder	Opret salgsordre	Salgsordre er blevet registreret i databasen	Find opret ordre i salgsdelsystemet Find kunde Tilføj kunde til salgsordren Vælg model Tilføj model til salgsordren (Vælg modul(er) Tilføj modul(er) til salgsordren) Færdiggør salgsordren

Salgsmedarbejder	Opret tilbud	Tilbud er blevet registreret i databasen	Find opret tilbud i salgsdel-systemet Find kunde Tilføj kunde til tilbuddet Vælg model Tilføj model til tilbuddet (Vælg modul(er) Tilføj modul(er) til tilbuddet) Færdiggør tilbuddet
Salgsmedarbejder	Skift tilbuds-type	Tilbudstype er blevet ændret	Find tilbud i salgsdelsystemet Vælg tilbud Skift tilbudstype Gem ændringer
Salgsmedarbejder	Annuler salgsordre	Salgsordre er blevet annulleret	Find salgsordre Vælg salgsordre Annullere salgsordre Gem ændringer
Lagermedarbejder	Opret vare	Vare er blevet registreret i databasen	Find opret vare i lagerdel-systemet Vælg leverandør Tilføj information om varen Færdiggør oprettelsen
Lagermedarbejder	Tilføj modul	Modul er blevet tilknyttet model	Find tilknyt vare i lagerdel-systemet Vælg ønsket model Tilføj modul til model Gem ændring
Lagermedarbejder	Opdater model	Model er blevet opdateret	Find opdater vare i lagerdelsystemet Vælg vare Tilføj ændringer Gem ændringer

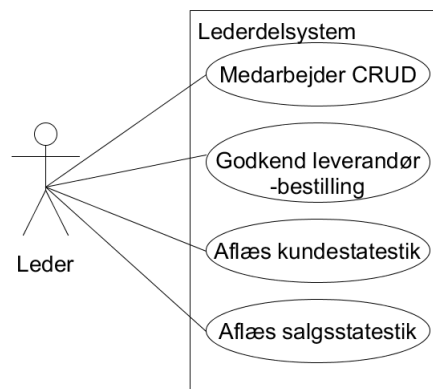
Lagermedarbejder	Opdater modul	Modul er blevet opdateret	Find opdater modul i lagerdelsystemet Vælg vare Vælg modul Tilføj ændringer Gem ændringer
Lagermedarbejder	Opdater leverandør	Leverandør er blevet opdateret	Find opdater leverandør i lagerdelsystemet Vælg leverandør Tilføj ændringer Gem ændringer
Lagermedarbejder	Registrer modtagelse	Salgsordre registreret modtaget	Find modtag salgsordre i lagerdelsystemet Vælg salgsordre Ændre salgsordre til modtaget Gem ændringer
Lagermedarbejder	Registrer levering	Status på salgsordre sættes til leveret	Find levering i salgsdelsystemet Vælg salgsordre Ændre salgsordrestatus til leveret Gem ændringer
Lagermedarbejder	Opret leverandør	Leverandør er blevet registreret i databasen	Find opret leverandør i lagerdelsystemet Tilføj informationer om leverandør Gem leverandør
Alle	Find model	Model er blevet fundet	Find søg model i lagerdelsystemet Søg efter model Vælg model
Alle	Find modul	Modul er blevet fundet	Find søg modul i lagerdelsystemet Søg efter modul Vælg modul

Alle	Find salgsordre	Salgsordre er blevet fundet	Find søg salgsordre i salgsdelsystemet Søg efter salgsordre Vælg salgsordre
------	-----------------	-----------------------------	---

Tabel 3.1: Medarbejder - opgave - mål tabel

### 3.4 Use-case diagrammer

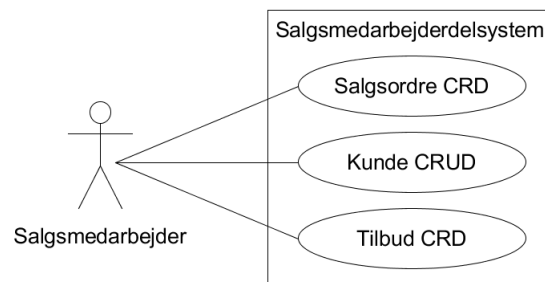
I dette afsnit opstilles use-case diagrammer ud fra metoden, der er beskrevet i bilag A.10.2. Disse udarbejdes for at identificere alle aktioner, en aktør kan eller skal kunne i samarbejde med en systemaktør.



Figur 3.3: use-case diagram for lederdelsystem.

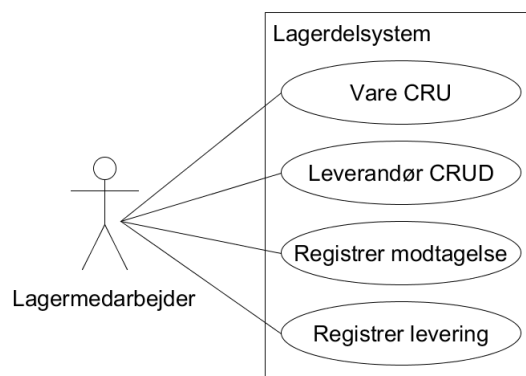
På figur 3.3 ses de use-cases som er forbeholdt en leder. Årsagen til, at en leder har use-casen Godkend leverandør bestilling er, at det er lederen, der skal godkende alle bestillinger, der oprettes i systemet. Det er også lederen, der håndterer oprettelsen af nye medarbejdere, og derfor har lederen 'medarbejder CRUD'. Desuden er det lederen der aflæser markedsføringssystemets genererede statistikker, for at træffe beslutning om hvor der skal markedsføres.





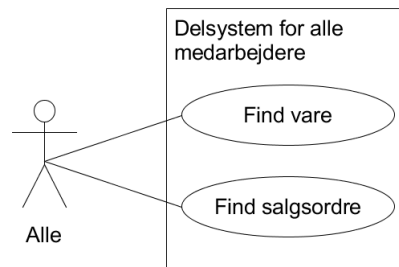
**Figur 3.4:** use-case diagram for salgsmedarbejdersystem.

På figur 3.4 ses use-casene salgsordre CRD, kunde CRUD og tilbud CRD som er use-cases for en salgsmedarbejder. Salgsmedarbejderen kan oprette, læse og annullere salgsordrer, hvilket forkortes til CRD (create, read delete). Til en salgsordren kan medarbejderen tilknytte en kunde, eller oprette en ny kunde, hvis kunden ikke allerede findes. Årsagen til, at et tilbud ikke kan opdateres skyldes, at det bliver til en salgsordre, hvis kunden accepterer det og ellers annulleres det.



**Figur 3.5:** use-case diagram for lagermedarbejdersystem.

Figur 3.5 illustrerer use-casene forbundet med en lagermedarbejder. Her er D'et er fjernet fra *CRUD*. Dette skyldes, at en vare ikke skal kunne fjernes igen, når den først har været i systemet. Det er desuden værd at bemærke, at vare dækker over både modeller og moduler i forhold til disse use-cases.



**Figur 3.6:** use-case diagram for medarbejderdelsystem.

Det sidste use-case diagram, som er vist på figur 3.6, illustrerer de use-cases alle medarbejdere kan have med systemet.

## 3.5 Use-case beskrivelser

I dette afsnit beskrives de mest komplekse use-cases, hvilke er blevet fundet ud fra afsnit 3.3. Metoden, som benyttes til udarbejdelse af disse brief use-cases, er beskrevet i bilag A.10.1.

### 3.5.1 Salgsmedarbejder

Use-case: Opret vare

En salgsmedarbejder ønsker at tilføje en ny vare til systemet. Salgsmedarbejdere benytter lager-delsystemet til at tilføje en ny vare med de relevante informationer. En vare har altid nogle bestemte attributter. Nogle varer er modificerbare, hvilket betyder, at der skal oprettes moduler til denne nye vare. Til sidst registreres den nye vare i systemet.

Use-case: Tilføj modul

En lagermedarbejder ønsker at tilføje et modul til en allerede eksisterende vare. Medarbejderen benytter lager-delsystemet til dette. Varen der ønskes tilføjet et nyt modul til, og herefter registreres det nye modul.

Use-case: Opret salgsordre

En salgsmedarbejder ønsker at oprette en ny salgsordre. Til dette benyttes IT-systemet. Salgsmedarbejderen tilføjer kunden og de ønskede varer til salgsordren som til sidst gemmes i databasen. Systemet opretter automatisk en leverandørbestilling af vare som ikke holdes på lager.

**Use-case: Skift tilbudstype**

En salgsmedarbejder ønsker at ændre typen af en salgsordre til at være leveret. Salgsmedarbejderen benytter IT-systemet til at finde salgsordren, efterfølgende ændrer salgsmedarbejderen tilbudstypen til at være leveret.

**Use-case: Opret kunde**

En ny kunde henvender sig til salgsmedarbejderen. Salgsmedarbejderen benytter IT-systemet til at indtaste information om den nye kunde. Salgsmedarbejderen gemmer den nye kunde i databasen.

**3.5.2 Leder****Use-case: Opdater medarbejder**

En leder ønsker at ændre information om en medarbejder. Lederen benytter IT-systemet til at finde den givne medarbejder, lederen ændrer medarbejderens information og gemmer ændringerne i databasen.

**3.5.3 Lagermedarbejder****Use-case: Registrer modtagelse**

En lagermedarbejder ønsker at registrer modtagelse af en salgsordre. Lagermedarbejderen benytter IT-systemet til at finde den givne salgsordre. Lagermedarbejderen ændrer salgsordrens type til at være modtaget. Lagermedarbejderen gemmer opdateringen i databasen.

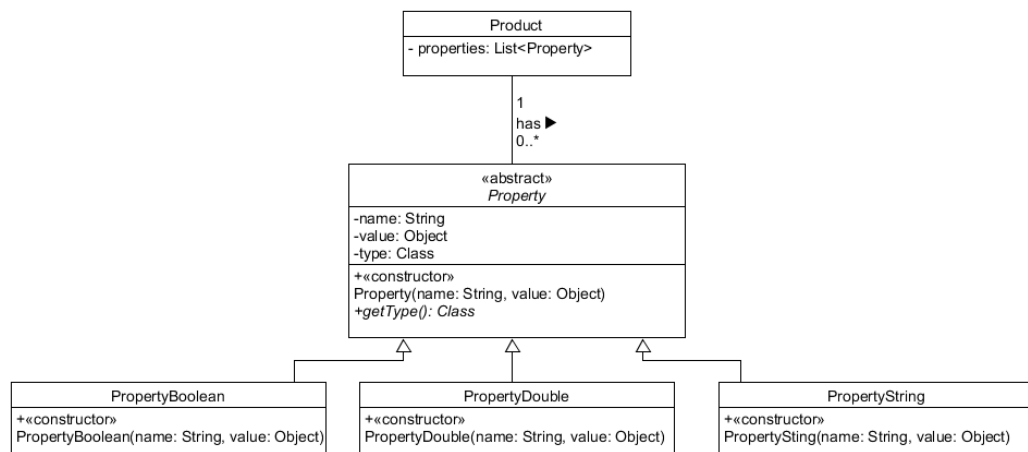
**Use-case: Registrer levering**

En lagermedarbejder ønsker at registre levering af en salgsordre. Lagermedarbejderen benytter IT-systemet til at finde den givne salgsordre. Lagermedarbejderen ændrer salgsordrens type til at være leveret. Lagermedarbejderen gemmer ændringerne i databasen.

**3.6 Proof of concept**

Ud over de opstillede krav udledt af interview og use-case modellering, udledes yderligt et funktionelt krav med hensyn til fremtidssikring af systemet i forbindelse med anskaffelse af nye produkter og sortimenter. En virksomhed som Brovst Bolighus udvider ofte sit vareudvalg med varer af forskellig karakter og attributter. Dette befordrer en forøget kompleksitet med henblik på håndtering af produkter.

En medarbejder skal kunne bruge systemet til at tilføje et nyt produkt eller produktkategori og tilskrive den sit eget sæt af attributter uden der skal laves en masse ændringer i systemet. Da dette vil kræve hidtil uprøvede designløsninger med hensyn til system-arkitekturen, har gruppen at undersøge om det kan lade sig gøre – et såkaldt proof og concept. Formålet med at lave et proof of concept er at afklare om der foreligger en løsning der tilfredsstiller givne tekniske systemkrav, hvor der i dette tilfælde er tale om arkitektur-signifikante krav. Efter noget research nåede gruppen frem til løsningen som vises på figur 3.7.



Figur 3.7: Diagram for implementation af property pattern.

Her gøres der brug af et designmønster kaldet property pattern, hvor produktet aggregerer en liste af Property-objekter. Property er en abstrakt superklasse som nedarver til de specifikke Property-klasser; PropertyString, PropertyDouble og PropertyBoolean. Disse nedarver alle Property's attributter; String name, Object value og Class type. Her er det værd at bemærke at den statiske type af value er Object, som gør at denne attributs dynamiske type kan sættes til samtlige af de tre Property-subklassers tiltænkte type. Den abstrakte metode getType() overrides i subklasserne som er sat til at returnere typen af den aktuelle subklassens value-type. Denne løsning har dog den begrænsning, at der kun kan tilføjes attributter af typerne String, Double og Boolean. Dog vurderes det at denne løsning er tilstrækkelig for Brovst Bolighus' behov.

Efter at have implementeret en fungerende prototype af dette designmønster er gruppen fortrøstningsfulde for at kunne imødekomme dette systemkrav.

### 3.6.1 Delkonklusion

Ud fra de mest komplekse use-cases, der er opstillet og beskrevet i dette afsnit, vil der i næste afsnit blive besluttet, hvilken use-case der prioriteres højest og samt beskrives grundlaget for beslutningen, hvorefter en fully dressed use-case præsenteres, og en domænemodel udarbejdes på baggrund af den valgte use-case.

## 3.7 Use-case prioritering

I dette afsnit bliver de forskellige use-cases prioriteret. Dette gøres for at finde ud af, hvilken use-case der bør implementeres først.

Use-cases	Kompleksitet	Dækningsgrad	Forretningsværd
Opret salgsordre	Høj	Høj	Høj
Find salgsordre	Lav	Mellem	Mellem
Annuller salgsordre	Lav	Lav	Lav
Opret tilbud	Høj	Høj	Mellem
Annuller tilbud	Lav	Lav	Lav
Godkend leverandørbestilling	Høj	Mellem	Høj
Kunde CRUD	Lav	Lav	Høj
Produkt CRU	Høj	Mellem	Høj
Leverandør CRUD	Mellem	Mellem	Høj
Registrer modtagelse	Mellem	Lav	Mellem
Registrer levering	Lav	Lav	Mellem
Læs salgsstatistik	Mellem	Høj	Høj
Læs kundestatistik	Høj	Mellem	Høj
Find vare på lager	Mellem	Lav	Lav

**Tabel 3.2:** Tabel over use-cases og prioritering.

Som det fremgår af tabel 3.2 så vurderes opret salgsordre til at have høj kompleksitet. Dette skyldes blandt andet at denne use-case kommer til at involvere mange forskellige klasser af systemet samt der er nogle forretningsregler som skal tages højde for. Ydermere har denne høj forretningsværdi som blev udledt af cost-benefit analysen i afsnit 2.6. Det ses også at Produkt CRUD har fået en højere kompleksitetsscore end normalt tillægges simple CRUD's. Dette skyldes den førømtalte øgede kompleksitet angående det omfattede udbud af vare af forskellig art med

unikke specialattributter, som står beskrevet i afsnit 3.6. Da implementationen af dette kommer til at involvere brug af uprøvet designarkitektur behæftes denne use-case med øget risiko og kompleksitet. En foreløbig iterationsplan for elaboration-fasen er baggrund af dette udarbejdet til følgende:

1. Opret salgsordre
2. Opret produkt

### 3.8 Use-case fully dressed

I dette afsnit beskrives nogle udvalgte use-cases på måden fully dressed. Metoden, til at beskrive en use-case fully dressed, er beskrevet i bilag A.10.1.

På tabel 3.3 beskrives use-casen opret salgsordre fully dressed.

Use-case	Opret salgsordre	
Aktør	Salgsmedarbejder	
Præbetingelse	Det ønskede produkt og tilhørende moduler eksisterer i systemet. Kunden findes i systemet. Der er forbindelse til databasen.	
Postbetingelse	Den komplette salgsordre er blevet tilføjet til databasen	
Frekvens	5 gange om dagen	
	Aktør	System
Flow of events	1. En kunde henvender sig for at bestille en vare	
	2. Medarbejderen påbegynder en ny salgsordre	3. Systemet opretter en ny salgsordre
	4. Medarbejderen finder kunden i systemet	5. Systemet fremviser information om kunden
	6. Medarbejderen vælger den ønskede kunde	7. Systemet associerer kunden til salgsordren
	8. Medarbejderen søger efter den ønskede model	9. Systemet fremviser moduler tilhørende modellen
	10. Medarbejderen søger efter det ønskede modul	11. Systemet viser det søgte modul
	12. Medarbejderen tilføjer det ønskede modul til salgsordren	13. Systemet associerer modulet til salgsordren og viser total
	14. Medarbejderen tilskrifter evt rabat til den pågældende ordrelinje	15. Systemet beregner og viser opdateret total

	<i>Trin 10-13 gentages til alle de ønskede moduler er tilføjet</i>	
	15. Medarbejderen afslutter salgsordren	16. Systemet tilføjer nødvendige attributter til salgordren
Alternative flow		17. Systemet gemmer salgsordren
		5a. Kunden findes ikke i systemet 1. Systemet prompter, at kunden skal oprettes
		9a. Varen findes ikke i systemet 1. Systemet viser en fejlmeddelelse
		11a. Modulet kunne ikke fremvises 1. Systemet viser en fejlmeddelelse
	12a. Medarbejderen har valgt et forkert modul 1. Medarbejderen vælger, hvilket modul der skal fjernes	2. Systemet fjerner det valgte modul fra salgsordren
	14a. Medarbejderen annullerer salgsordren	1. Systemet nulstiller de attributter, som er blevet føjet til salgsordren
		15a. Systemet kunne ikke tilføje de nødvendige attributter 1. Systemet viser en fejlmeddelelse
		16a. Systemet kunne ikke tilskrive salgsordren til databasen 1. Systemet viser en fejlmeddelelse 2. Systemet gemmer salgsordren i en lokal fil 3. Systemet forsøger at forbinde og tilskrive salgsordren til databasen

**Tabel 3.3:** Fully dressed beskrivelse for use-casen opret salgsordre.

Nogle supplerende krav til use-casen opret salgsordre er følgende:

1. Bruger kan udvælge kunde og vare fra en fremvist liste
2. Rabatsats kan ikke overstige delttotal (100 %)
3. Rabatsats kan ikke være under nul
4. Brugeren skal altid kunne afbryde sin nuværende handling
5. I tilfælde af fejl skal kunden have en forklarende fejlbeskrivelse
6. Brugeren skal ikke være i tvivl om ord og handlingers betydning

På tabel 3.4 beskrives use-casen opret produkt på formen fully dressed.

Use-case	Opret produkt	
Aktør	Lagermedarbejder	
Præbetingelse	Der er forbindelse til databasen. Produkttypen for det produkt, der ønskes oprettet, eksisterer i databasen. Leverandøren eksisterer i databasen	
Postbetingelse	Produktet er blevet tilføjet til databasen med korrekte attributter	
Frekvens	2 gange om ugen	
	Aktør	System
Flow of events	1. Der er kommet en ny vare hjem, som ikke er oprettet i systemet	
	2. Medarbejderen klikker på opret produkt i systemet	3. Systemet finder eksisterende produkttyper fra databasen.
	4. Medarbejderen vælger den produkttype, som det nye produkt skal oprettes under.	5. Systemet fremviser felter til nødvendigt information
	6. Medarbejderen udfylder informationer om det nye produkt	
	7. Medarbejderen vælger leverandør.	
	8. Hvis den valgte produkttype er en modul-type vælger medarbejderen den specialdesignvare modulen skal forbindes med.	
	9. Medarbejderen klikker på opret produkt.	9. Systemet associerer det nye produkt med valgte leverandør.
		1. Systemet associerer det nye produkt med valgte produkttype
		12. Systemet tilføjer nødvendige attributter til produktet



		13. Systemet gemmer produktet
Alternative flow		3a. Der er ikke forbindelse til databasen 1. Systemet viser en fejlmeddelelse
		9a. Medarbejderen har ikke udfyldt alle felterne 1. Systemet giver en fejlmeddelelse

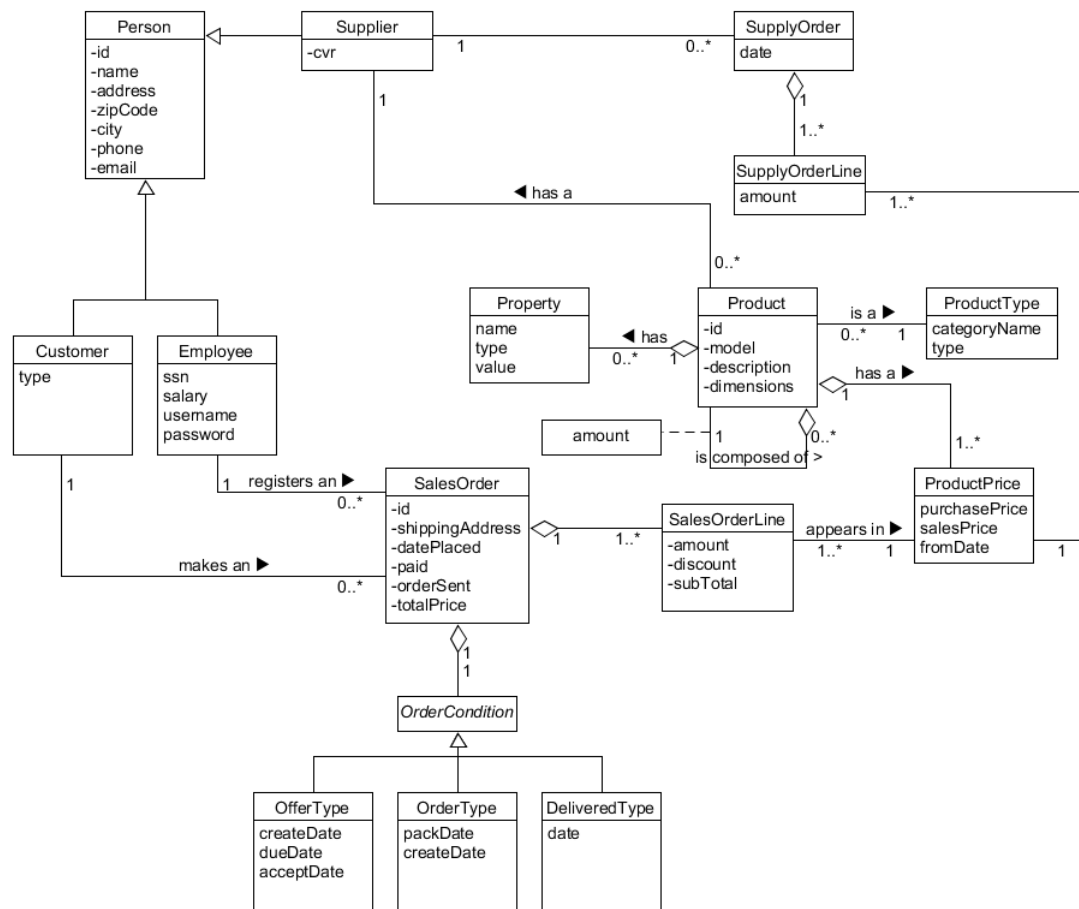
**Tabel 3.4:** Fully dressed beskrivelse for use-casen opret produkt.

De supplerende krav til denne use-case er:

1. Bruger kan udvælge mulige produkttyper fra en fremvist liste
2. Brugerens skal altid kunne afbryde sin nuværende handling
3. I tilfælde af fejl skal kunden have en forklarende fejlbeskrivelse
4. Brugerens skal ikke være i tvivl om ord og handlingers betydning

### 3.9 Domænemodel

I dette afsnit gennemgås og forklares domænemodellen for SmartOrder. Metoden til at udvikle domænemodeller står beskrevet i bilag A.11.



Figur 3.8: Domænemodel for systemet til Brovst Bolighus.

Figur 3.8 viser domænemodellen for Brovst Bolighus. Af domænemodellen forekommer der konceptuelle klasser, som skal benyttes som byggesten til systemet.

I SmartOrder fungerer kunder, ansatte og leverandører som personer. Dette er illustreret ved at klasserne Customer, Employee og Supplier alle er specialisering af klassen Person. Dette skyldes at klasserne deler mange attributter. Det ses endvidere, at Customer og Employee klasserne har en 'nul-til-mange' relation til klassen SalesOrder og grunden til dette er, at der altid skal en ansat til at lave en salgsordre, og den samme ansatte kan lave flere salgsordre. Det samme er gældende for en kunde, da kunden kan købe over flere omgange. Dette begrænser dog også systemet til at hvis en ansat gerne vil bestille en vare, er denne nød til også at skulle oprettes som kunde.

Product klassen er en central del af systemet, og dette er symboliseret ved de man-

ge forbindelser klassen har til andre klasser. Product klassen har en ProductPrice-associering, da det er nødvendigt altid at vide hvad der betales for en vare, og hvad den vare sælges for. Product har også en ProductType-associering da vi skal vide, om et produkt er en sofa, stol eller spisebord. ProductType-klassens type-attribut indikerer en yderlig inddeling af produkttyper i henholdsvis hyldevare, specialdesignvare eller modul til specialdesignvare. Da et Product kan have moduler, men et modul samtidigt også er et Product gøres der brug af den rekursive *composite pattern*, hvor Product associerer sig selv. Der findes en Property-associering da et produkt kan være mange ting, hver med et unikt sæt specialattributter. Product kan således have én til mange reference til Property afhængigt af behovet for specialattributter. Et produkt har også en Supplier-associering, da produktet bliver indkøbt fra en leverandør.

For at knytte systemet sammen findes SalesOrder-klassen, der er ment til at holde styr på, hvad der bliver solgt. OrderCondition aggregerer til SalesOrder og beskriver, hvilken status en salgsordre har. Dette ses ved det trinvis rollemønster, hvor OrderCondition er en abstrakt klasse, som har tre subklasser. SalesOrderLine aggregerer til SalesOrder, da en salgsordre består af flere salgsordrerlinjer.

### 3.10 Konklusion på inceptionfasen

Med use-casen opret salgsordre og opret vare in mente og en udarbejdet domæne-model for det samlede system viser det sig, at systemet bevæger sig imod en ERP (Enterprise resource planning) model. Derfor vil næste kapitel forsøge at implementere en datalogisk IT-løsning på de givne use-case, med tanken om, at Brovst Bolighus skal kunne benytte dette system til deres daglige opgaver. Det er håbet, at den datalogiske løsning ville kunne fungere som et komplet program, til at hjælpe virksomheden med alle aspekter af daglig styring samt på et senere tidspunkt opfylde alle deres ønsker om IT-systemer, og derved samle alt i ét.



## 4 | Elaboration

Elaboration-fasen bygger videre på inception-fasen og her udvikles system-sekvensdiagrammer, operationskontrakter, kommunikationsdiagrammer, designklassediagrammer. Derudover præsenteres der også kode, som er blevet implementeret i forbindelse med udvikling af systemet.

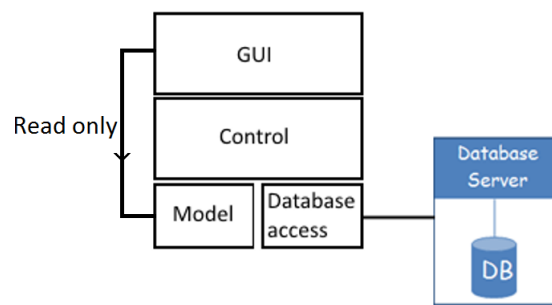
### 4.1 Iteration 1 - Opret salgsordre

I dette afsnit beskrives den første iteration der blev udviklet, som er opret salgsordre. Den overordnede arkitektur for systemet vil også blive beskrevet, da den allerede delvist skal udvikles samtidig med den første iteration.

#### 4.1.1 Arkitektur

I dette underafsnit vil der beskrives nogle patterns og vises kodeeksempler på, hvordan disse patterns er blevet implementeret. De patterns, der bliver beskrevet, er *GRASP*, *Strategy pattern* og *Composite design pattern*.

Arkitekturen i et system definerer, hvordan systemets forskellige dele interagerer med hinanden. Ønsket er at opnå en arkitektur med lav kobling og høj samhørighed - hvilket betyder, at systemets dele er adskilt så vidt muligt og afhænger ganske lidt af andre dele. Derudover skal de enkelte dele have overskuelige og let forståelige ansvarsområder. Den hyppigst brugte arkitektur er tre-lags arkitekturen, som inddeler systemet i tre dele: grænseflade-, controller- og modellag. Denne type arkitektur er delt i to slags, henholdsvis åben og lukket arkitektur. Den åbne tillader læsning fra alle underliggende lag. En lukket arkitektur tillader kun læsning fra det nærmeste underliggende lag. Det betyder, at grænsefladelaget skal gennem controllerlaget, før det kan tilgå model-laget.



**Figur 4.1:** Eksempel på en tre-lags åben arkitektur med database [5].

Den arkitektur, der benyttes til at udvikle SmartOrder, er en åben tre-lagsarkitektur, som er illustreret på figur 4.1.

#### 4.1.2 Database

Som nævnt i afsnit 4.1.1 bliver der benyttet en arkitektur som gør brug af en database. Den database, som er benyttet til dette projekt, er en SQL-server. Den relationelle model er udarbejdet ud fra domænemodellen, som er beskrevet i afsnit 3.9. Måden, hvorpå en domænemodel laves om til en relationel model, er, at hver klasse afbildes over i en tabel, hvor det derefter overvejes, hvorvidt tabellen bør udbygges med en kolonne, som indeholder en entydig reference til ethvert objekt [6].

Et mål når en tabel designes, er at den skal være nem at forstå. Dette kan gøres ved at designe tabellen sådan at, der ikke anbringes forskelligt data i samme tabel. Et andet mål er at undgå redundans, da det øger pladsforbruget og kan give problemer ved opdatering af data. Derudover bør antallet af NULL-værdier minimeres, da det også øger pladsforbruget og kan give flere fortolkninger. Uægte tupler bør også undgås, da den samme person kan komme til at fremtræde forskellige steder. Selvom det ikke giver det kartetiske produkt, giver det mange dubletter og ikke et korrekt billede af hvordan databasen egentlig burde være [6].

Til at modellere databasen er der nogle retningslinjer som bør følges. Associeringer og aggregeringer transformeres til fremmednøglerreferencer hvorom det gælder [6]:

- 1-1: Inkluder den ene sides primærnøgle som fremmednøgle på den anden
- 1-n: 1-sidens primærnøgle skal være fremmednøgle på n-siden
- n-m: Der laves en ny tabel med begge siders primærnøgler som fremmednøgler

Måden, hvorpå generaliseringer afbildes i tabeller skal være en af følgende [6]:

1. Hver klasse afbildes i en tabel. Her bindes de generelle og specielle dele af et objekt sammen med nøgler.
2. Hver specialiseringsklasse har sin egen tabel, som også indeholder generaliseringsklassens attributter.
3. Generaliseringsklassen afbildes i en tabel, som også indeholder alle specialiseringsklassernes attributter.

Når en database modelleres gøres der brug af normalformerne, som er en formel formulering af designmål for tabeller. Der findes i alt seks normalformer (NF). I denne rapport tages der dog kun højde for de fire første.

En tabel er første normalform hvis der kun er simple attributter, hvilket vil sige, at der ikke er nogle sammensatte eller flereværdi attributter. Denne form sikres ved transformation fra domænemodel til den relationelle model.

Anden normalform forholder sig til partielle afhængigheder. En fuldt funktionel afhængighed er, hvis det ikke kan lade sig gøre at fjerne nogle attributter fra "X" uden "X" → "Y" ophæves. Hvis en funktionel afhængighed ikke er fuld er den partiel. Denne form forekommer hvis tabellen er på 1. NF og alle ikke-nøgle attributter er fuldt afhængige af kandidat-nøglerne.

Den tredje normalform forholder sig til transitive afhængigheder, hvilket vil sige, at hvis der eksisterer nogle attributter, fx "B", hvorom det gælder at "A" → "B" og "B" → "C". En tabel er på 3. NF hvis den er på 2. NF og ingen ikke-nøgle attributter er transitivt afhængige af en kandidat-nøgle.

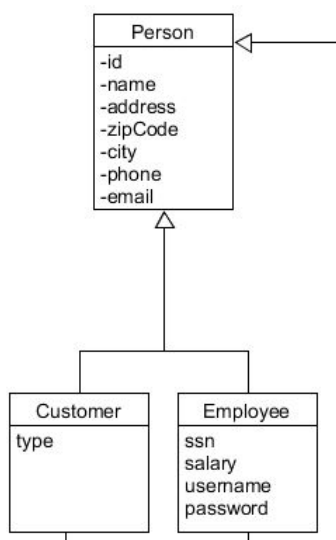
En tabel er på BCNF hvis der for alle funktionelle afhængigheder gælder at determinanten er en super-nøgle. Det vil sige at attributterne i en tabel skal afhænge af nøglen, hele nøglen og intet andet end nøglen. Hvis en tabel er på BCNF er den også på 1., 2. og 3. NF [7].

Til design af Property-tabellen er tredje generaliseringsregel benyttet. Da det er begrænset hvor mange attributter der skal gemmes, er det mest optimale at lave en tabel med en enkel tilgang. Derudover kan der argumenteres for hvorvidt antallet af NULL-værdier fylder i databasen, da systemet ikke er større. Designet af Property-tabellen er vist på figur 4.2.

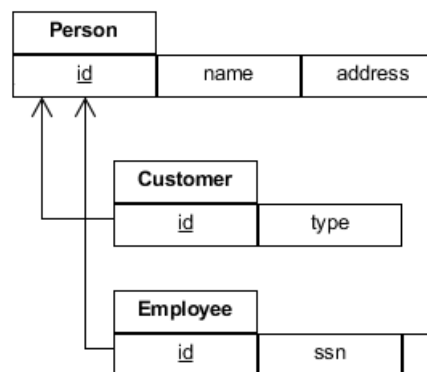
Property					
<u>id</u>	name	string_value	double_value	boolean_value	<u>product_id</u>

**Figur 4.2:** Her ses property tabellen i den relationellemodel, som afspejler databasen.

For hver tabel er der implementeret en id kolonne, som autogenereres. Som det ses på figur 4.3 og 4.4, har customer og employee en fremmednøglerreference til persontabellens id, da customer og employee er specialiseringer af person. Dette viser at der er gjort brug af første generaliseringsregel.



**Figur 4.3:** Viser domænemodellen hvor customer og employee arver attributter fra Person.



**Figur 4.4:** Viser den relationelle model hvor employee og customer tabellerne har fået Person klassens autogenerede id attribut som fremmednøgle.

Phone-attributten fra Person-klassen fået sin egen tabel, da en person ofte har mere end et telefonnummer. Tabellen kan derved gemme flere forskellige telefonnumre med det samme personid, hvilket er et eksempel på 2. NF. For at overholde BCNF er City også flyttet væk fra Person-tabellen, da det ikke er person-tabellen der fortæller noget om byen, men derimod postnummeret.

Samlet set er der et mindre brud på 1. NF, da "name" indeholder både for- og efternavn, hvilket giver tuplen en sammensat værdi. Årsagen til at det er lavet på denne måde er, at det ikke umiddelbart vil give mening at splitte for- og efternavn i dette system. Scriptet der bruges til at oprette den aktuelle database kan findes i bilag J.



## GRASP

GRASP står for General Responsibility Assignment Software Patterns og bruges til at beskrive fundamentale principper i design af objekter og tildeling af ansvar. Formålet med at bruge GRASP-mønstret er at gøre systemet lettere at forstå, vedligeholde og genbruge.

De forskellige mønstre der er i GRASP er *controller*, *information expert*, *creator*, *low coupling* og *high cohesion*.

Controller-mønstret bruges til at fremme sandsynligheden for at de forskellige use-cases holdes adskilt fra grænsefladen. Dette giver samtidig mulighed for at ændre og genbruge kode. Der benyttes i dette system en controller per use-case. Dette giver lav kobling og høj samhørighed, da controllerne ikke er så afhængige af hinanden.

Creator-mønstret bruges til at finde ud af, hvilket objekt der skal have ansvaret for at oprette en ny instans af en klasse. I SmartOrder benyttes løsningen, hvor en klasse har de initierende data for en anden klasse. Det ses eksempelvis mellem SalesOrder og SalesOrderLine, hvor SalesOrder-klassen står for at oprette SalesOrderLine-objekter hvilket er vist på figur 4.6. Dette gør, at der opnås høj samhørighed men også høj kobling, da SalesOrder står for tilføjes af SalesOrderLines, og SalesOrderLines oprettes i klassen SalesOrder.

Information expert bruges til at finde ud af hvilken klasse der skal stå for at oprette objekterne. Når der skal oprettes og skrives nye objekter til databasen bruges controller-klasserne. Hvis der skal bruges nogle objekter i systemet bruges database-laget til at bygge disse objekter.

## Strategy pattern

Ved brug af strategy pattern, kan en klasse eller dens algoritme ændre egenskaber i run-time, altså mens programmet kører.

Det betyder, at der oprettes instanser af klasser, der har et strategy pattern og en eller flere hjælpe klasser, også kendt som context classes, der under run-time ændrer opførsel afhængig af hvilken opgave der skal løses.

Implementationen af strategy pattern laves ved at subklasser nedarver fra en abstrakt klasse, eller et interface. Som det fremgår i domænemodellen vises DiscountIF som en attribut i klassen SalesOrderLine. Interfacet der benyttes er vist på li-

sting 4.1.

```
1 public interface DiscountIF {  
2     double calculateDiscount(String amount);  
3     double getDiscount();  
4 }
```

**Listing 4.1:** DiscountIF-interfacet

Discount er blevet implementeret som et strategy pattern. Mønstret er implementeret således at interfaceet DiscountIF har tre subklasser: DiscountByProcent(), DiscountTemplate() og DiscountByAmount().

```
1 public void setDiscount(String discountAmount) {  
2     if(discountAmount.contains("%")){  
3         discountPattern = new DiscountByProcent(discountAmount,  
4             product);  
5         discount = (discountPattern.getDiscount() * amount);  
6     }  
7     else if(discountAmount.isEmpty()){  
8         discountPattern = new DiscountTemplate(discountAmount);  
9     }  
10    else{  
11        discountPattern = new DiscountByAmount(discountAmount,  
12            product);  
13        discount = discountPattern.getDiscount();  
14    }  
15 }
```

**Listing 4.2:** Metoden "setDiscount", som bruges til at angive rabat

I metoden setDiscount, som er vist på listing 4.2, tages en String som parameter, hvilket gøres for at finde ud af, hvilken form for rabat en medarbejder vil give. Herefter instansieres subklasserne til attributten discountPattern som er af typen DiscountIF, efterfølgende sættes attributten discount til at være den værdi som gives i discountPattern.

## Composite Design Pattern

I et composite pattern findes der composite objekter, hvilket betyder at de indeholder andre objekter. Composite pattern går ud på, at begge typer objekter kan manipuleres på nøjagtig samme måde, uden der nødvendigvis kendes forskel på dem.

Som det ses på domænemodellen, figur 3.8, kan et Product have nul til mange Products tilhørende. Dette ses ved en rekursiv aggregering og bliver implementeret således et Product har en LinkedList bestående af Products. Implementationen er vist på listing 4.3.

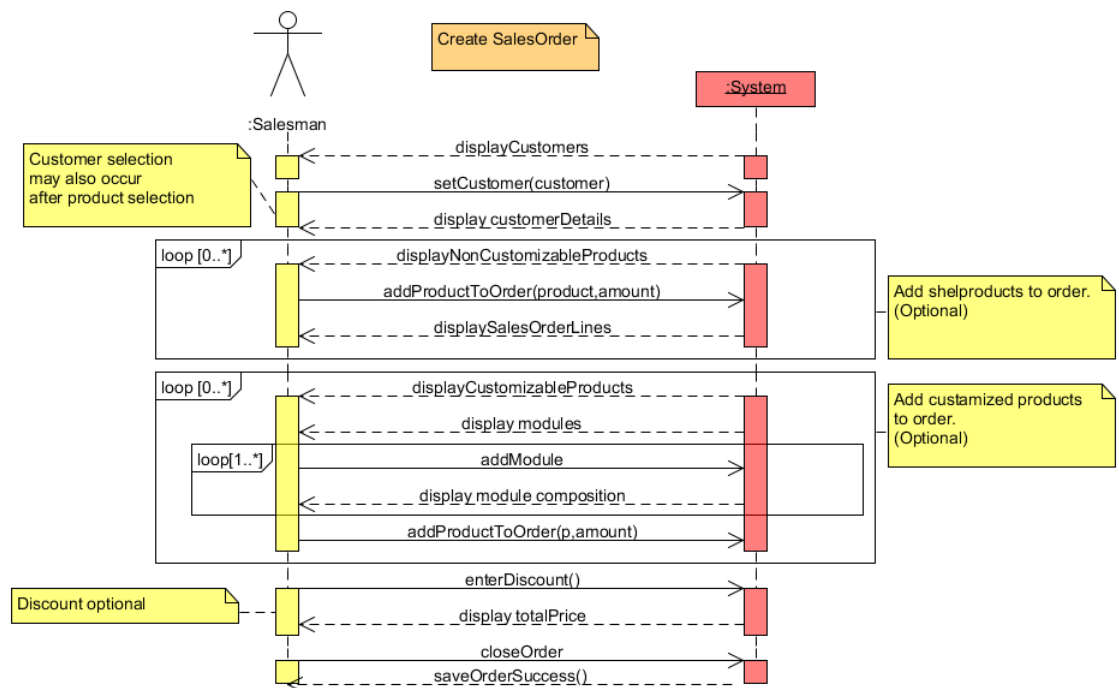
```
1 public class Product implements ProductIF {  
2     ...  
3     private LinkedList<Product> modules;  
4     ...  
5     public void addModuleToProduct(Product p) {  
6         modules.add(p);  
7     }  
8     ...
```

**Listing 4.3:** Implementation af Composite Design Pattern.

Dette gøres fordi et Product kan have flere moduler tilknyttet sig, men et modul er også et Product.

### 4.1.3 System-sekvensdiagram

På figur 4.5 illustreres system-sekvensdiagrammet for use-casen opret salgsordre, hvoraf det fremgår, hvilke metoder der kaldes og metodekaldenes returtype, når en bruger sender et bestemt input til systemet. I overensstemmelse med kravspecifikationerne beskrevet i afsnit 3.3, ønsker brugeren at vælge kunde og varer ud fra en fremvist liste. Hvert trin starter således med at systemet præsenterer brugeren med lister, hvorefter brugeren søger og finder de ønskede objekter. Rækkefølgen for tilføjelsen af kunde og varer er uvilkårlig, og varer ville således kunne vælges først. Som det fremgår af de to loops er disse trin frivillige. Brugeren kan nøjes med at oprette en salgsordre for en hylde-vare (*nonCustomizable*) uden at skulle tilføje en specialdesignet vare (*customizable*) og omvendt.



**Figur 4.5:** Systemsekvens-diagrammet for use-casen opret salgsordre.

#### 4.1.4 Operationskontrakt

I dette afsnit udarbejdes en operationskontrakt på use-casen opret salgsordre. Dette gøres for at give et detaljeret overblik over de forskellige objekters stadier ved udførelse af operationen `closeOrder()`. Metoden, som benyttes til at udarbejde en operationskontrakt, er beskrevet i bilag A.12

<b>Operation: closeOrder()</b>
Use-case: opret salgsordre <b>Præbetingelse:</b> <ul style="list-style-type: none"> <li>- En instans <i>so</i> af typen <i>SalesOrder</i> er blevet oprettet.</li> <li>- Én til mange instanser af <i>SalesOrderLine</i>, <i>salesOrderLines</i>, er associeret til <i>so</i>.</li> <li>- Én til mange instanser <i>p</i> af typen <i>Product</i> er tilknyttet <i>salesOrderLines</i>.</li> <li>- <i>salesOrderLine.get(i).amount</i>, <i>salesOrderLines.get(i).discount</i> og <i>salesOrderLines.get(i).subTotal</i> er tilskrevet en værdi</li> <li>- En instans <i>c</i> af typen <i>Customer</i> er associeret til <i>so</i>.</li> </ul> <b>Postbetingelse:</b> <ul style="list-style-type: none"> <li>- En instans <i>oc</i> af typen <i>OrderCondition</i> blev oprettet og associeret til <i>so</i></li> <li>- <i>so</i>, <i>oc</i> og alle instanser af <i>SalesOrderLine</i> blev gemt i databasen</li> </ul>

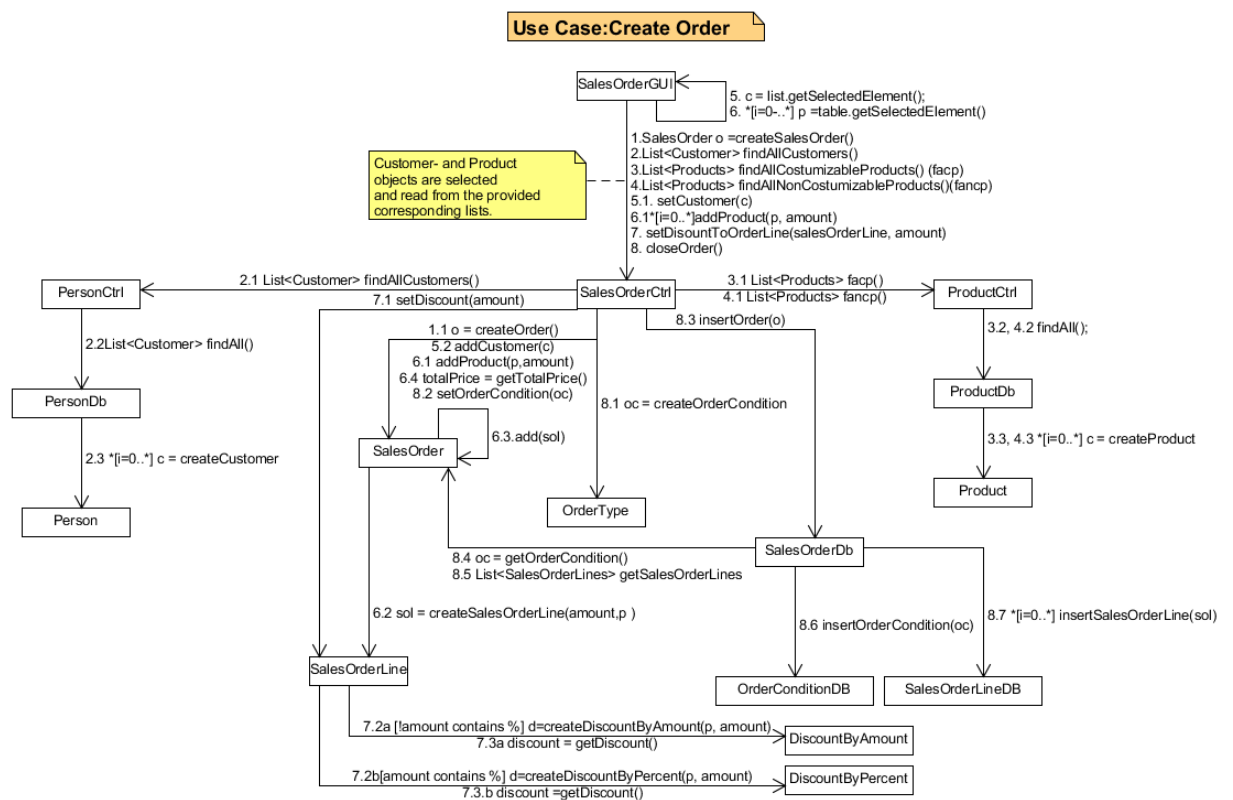
Tabel 4.1: Operationskontrakt af use-casen opret salgsordre

På tabel 4.1 ses operationskontrakten for `closeOrder()`. Her vises både præ- og postbetingelser og hvilke klasser og attributter som indgår i denne use-case. Især i præbetingelserne ses de mange klasser og deres mange associeringer som skal oprettes og gemmes i databasen.

#### 4.1.5 Kommunikationsdiagram

I dette afsnit opstilles et kommunikationsdiagram. Kommunikationsdiagrammer bruges til at illustrere hvilke metoder de forskellige objekter gør brug af [8, s.197]

Figur 4.6 viser kommunikationsdiagrammet der er udviklet til use-casen opret salgsordre.

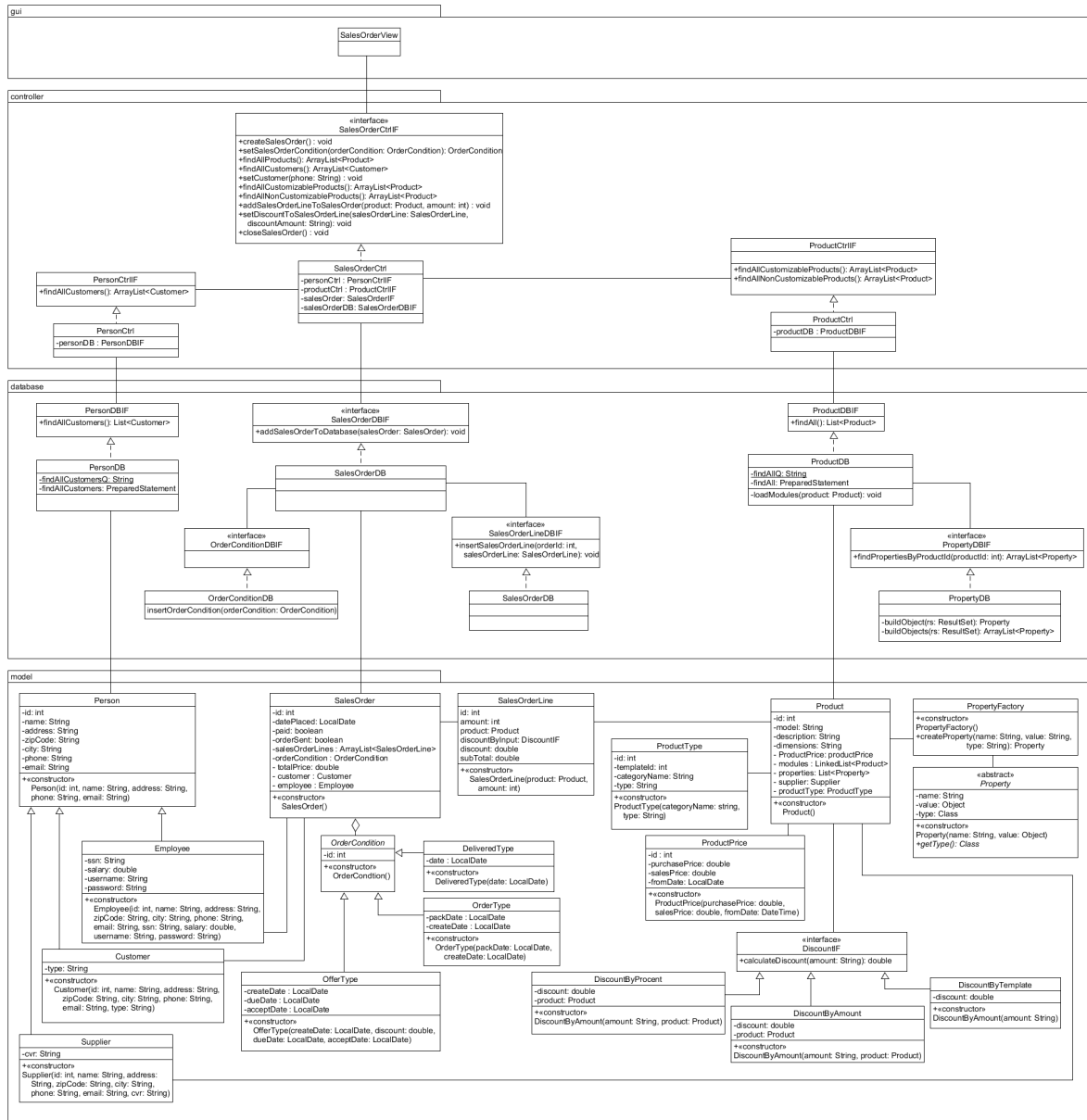


Figur 4.6: Kommunikationsdiagram for use-casen opret salgsordre.

I overensstemmelse med systemsekvens-diagrammet vist på figur 4.5 hentes der lister op med kunder `findAllCustomers()`, hyldevarer `findAllNonCustomizableProducts()` og specialdesignede varer `findAllCustomizableProduct()`. Brugeren udvælger de ønskede objekter fra listerne, og disse tilknyttes henholdsvis `SalesOrder`-objektet og `SalesOrderLine`-objekterne i trin 5 og 6. Brugen af det førnævnte Strategy Pattern ses i trin 7.1 og 7.2 i forbindelse med tilskrivelse af `Discount`. Metoden `setDiscount(amount)` vil ud fra `amount` instantiere den `DiscountIF`-klasse, som har den rette beregningsalgoritme. Benyttelsen af det førnævnte GRASP patterns fremgår også af diagrammet. I overensstemmelse med *Creator*-mønsteret ser vi i trin 6.2, at `SalesOrder` har ansvaret for oprette `SalesOrderLine`-objekter da disse er tæt knyttet via aggregering. Brug af *Information Expert*-mønsteret ses i trin 7.1, hvor den klasse der har den nødvendige information, `SalesOrderCtrl`, sætter `SalesOrderLines` discount attribut.

### 4.1.6 Designklassediagram

Efter foregående designmodellering udvindes et endelig designklassediagram, som vil være basis for implementeringen af use-casen Opret salgsordre. Dette diagram er vist på figur 4.7.



Figur 4.7: Designklassediagram for use-casen opret salgsordre.

## 4.2 Iteration 2 - Opret produkt

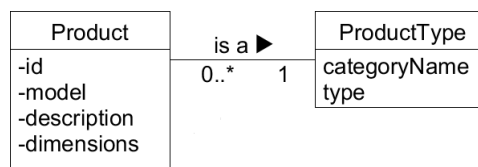
Den foregående iteration har allerede behandlet dele af produkt CRUD navnlig read, da produkter bliver hentet op, læst og tilknyttet en salgsordre. Denne iteration vil have til formål yderlig at implementere denne arkitektur ved bearbejde opret produkt.

### 4.2.1 Arkitektur

I dette underafsnit vil der blive beskrevet de forskellige patterns som er blevet benyttet til udarbejdelse af use-casen opret produkt.

#### Type Pattern, Property Pattern og Prototype Patten

Et af de tidlige problemer der opstod ved modelleringen af de konceptuelle klasser i domænemodellen var et u hensigtsmæssig stort antal af subklasser, da virksomheden har et uoverskuelig varesortiment og antal varegrupper. Til at løse dette problem er der gjort brug af et mønster som kaldes Type pattern, hvor alle subklasserne transformeres til én Type-klasse. Implementationen heraf er illustreret på figur 4.8 og listing 4.4.



Figur 4.8: Implementation af Type Pattern

```

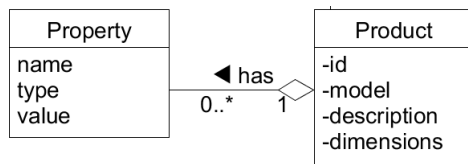
1 public class Product {
2     private ProductType productType;
3 }
  
```

Listing 4.4: Implementation af Type-Pattern

Et andet problem er at fremtidssikre systemet med henblik på udvidelse af varesortimentet. En virksomhed som Brovst Bolighus vil fra tid til anden diversificere sig og afprøve salg af nye varegrupper. En ny varetype medfører ofte nye unikke sæt attributter. F.eks. vil salg af brugskunst kræve en kunstner, hvorimod en sofa kun har en producent. Der foreligger således et funktionelt krav om, at Product kan

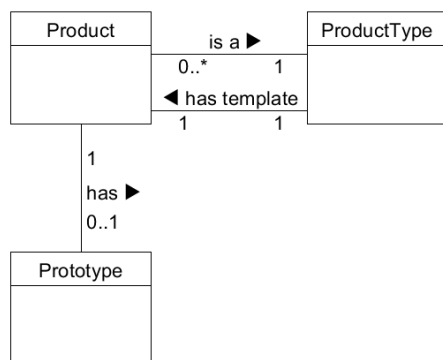


have en samling af specialattributter. Dette problem løses ved brug af **Property Pattern** som er illustreret på figur 4.9.



Figur 4.9: Implementation af Property Pattern

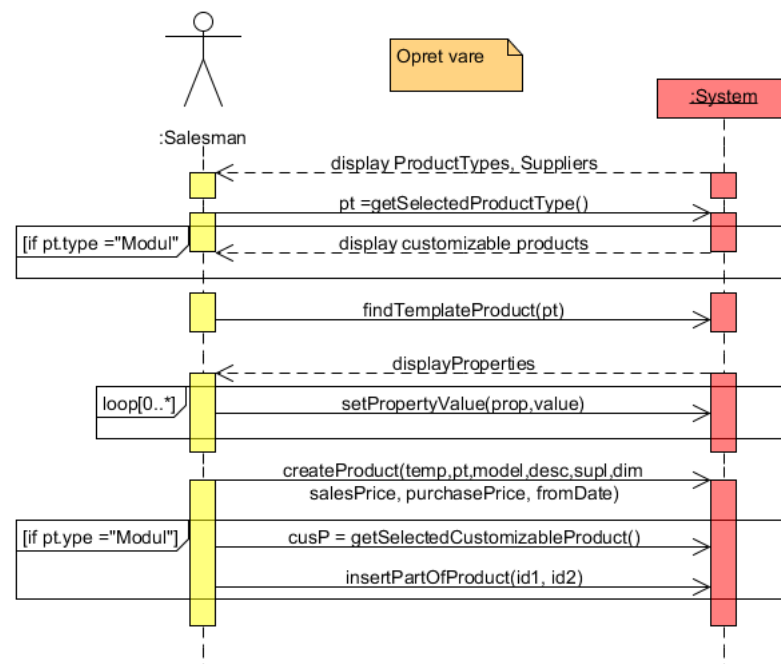
Umiddelbart vil det virke mere hensigtsmæssigt, at Property associeres med ProductType da en specialattribut deles af samtlige varer inden for samme type. Grunden til, at Property i stedet associeres med Product, skyldes brugen af **Prototype Pattern**. I stedet for at ProductType bærer direkte reference til Property, associeres der til et *prototype*-product som er en skabelon for alle nye produkter af samme type. Prototypen er således et tomt Product-objekt gemt i databasen som bærer reference til de Property-objekter som skal laves sammen med nye produkter. Hvis der eksempelvis skal tilføjes en hængekøje til sortimentet, hentes dennes prototype op fra databasen, de relevante felter udfyldes, produktet "klones" og indsættes i databasen. På figur 4.10 ses et samlet diagram for de tre designmønstre. Bemærk den dobbelte associering mellem Product og ProductType som viser at produkt er af en bestemt type, men at produkttype har en skabelon til nye produkter.



Figur 4.10: Kombination af Type-, Property- og Prototype Pattern

## 4.2.2 System-sekvensdiagram

I dette afsnit vises og beskrives system-sekvensdiagrammet til use-casen opret produkt.



Figur 4.11: Systemsekvens-diagram for opret produkt

På figur 4.11 vises system-sekvensdiagrammet for use-case opret produkt. I overensstemmelse med kravspecifikationen vil brugeren få fremvist lister til udvælgelse af produkttype og leverandør. Hvis den valgte produkttype er af typen "modul" vil der fremvises yderligere en liste af specialdesignede produkter. Herfra vælger brugeren, hvilket af produkterne produktet skal tilhøre. Systemet viser en tabel af specialattributter som tilhører den valgte produkttype. Disse og de resterende attributter tilskrives værdier af brugeren som systemet bruger til oprette og gemme de rette objekter.

### 4.2.3 Operationskontrakt

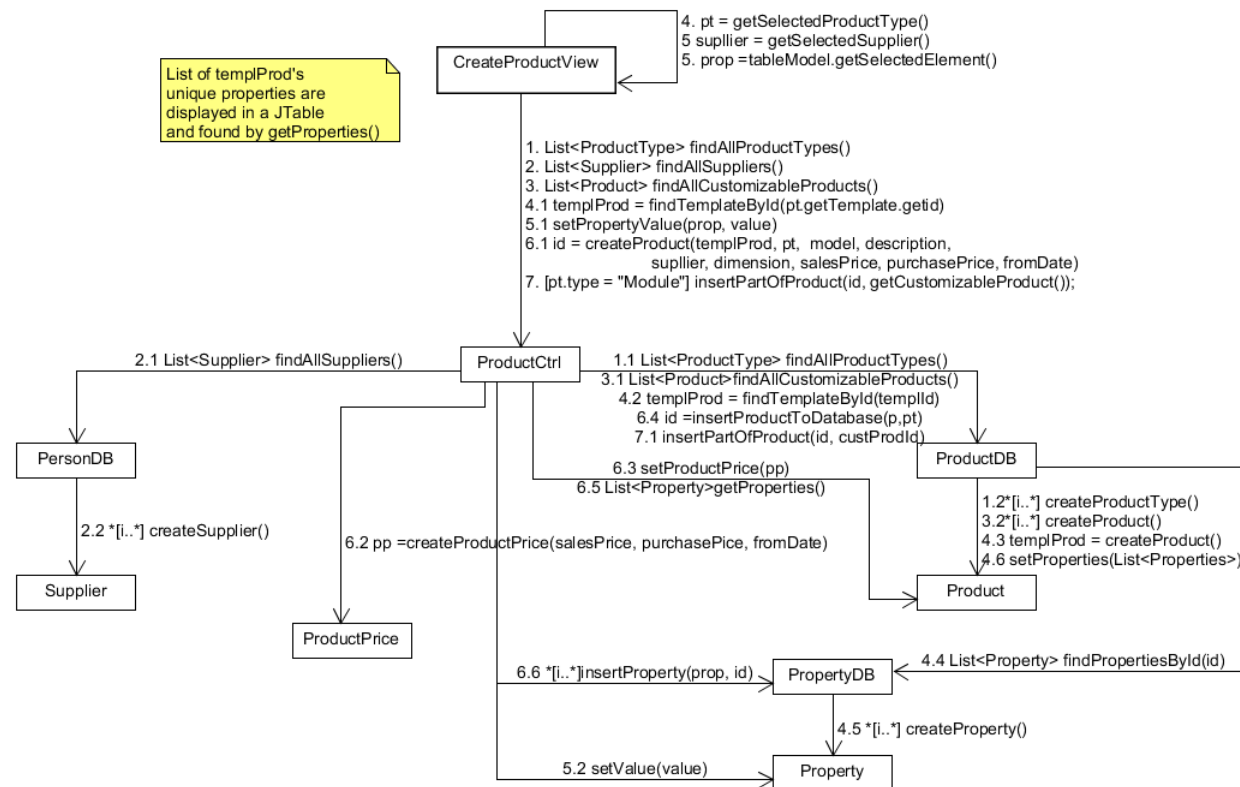
På tabel 4.2 ses operationskontrakten for operationen `createProduct(...)`. Her illustreres, hvorledes de tiltænkte designmønstre opererer i forbindelse med use-casen. Skabelon-objektet (*tempP*) tilhørende det valgte *ProductType*-objekt *pt*, hentes og instantieres. Derefter instantieres en liste af *Property*-objekter som bærer reference til *tempP*. Attributterne tilskrives værdier ud fra metodens parameterliste.

<b>Operation: createProduct(tempP, pt, model, description, supplier, dimension, salesPrice, purchasePrice, fromDate)</b>
Use-case: opret product <b>Præbetingelse:</b> - En instans <i>pt</i> af typen <i>ProduktType</i> er oprettet. - En instans <i>sup</i> af typen <i>Supplier</i> er blevet valgt fra en fremvist liste. - En instans <i>tempP</i> af typen <i>Product</i> og en liste, <i>properties</i> , af af associerede objekter af typen <i>Property</i> er blevet oprettet fra databasen. <b>Postbetingelse:</b> - <i>tempP.model</i> , <i>tempP.description</i> , <i>tempP.supplier</i> . <i>tempP.dimensions</i> blev tilskrevet værdier. - <i>properties.get(i).value</i> blev tilskrevet værdier - En instans <i>pp</i> af typen <i>ProductPrice</i> blev oprettet. - <i>pp.salesPrice</i> , <i>pp.purchasePrice</i> og <i>pp.fromDate</i> blev tilskrevet værdier. - <i>tempP.productPrice</i> blev sat til <i>pp</i> . - <i>tempP</i> blev gemt i databasen under et nyt id. - <i>pp</i> , <i>pt</i> , <i>properties</i> blev gemt i databasen.

Tabel 4.2: Operationskontrakt af use-casen opret produkt

#### 4.2.4 Kommunikationsdiagrammer

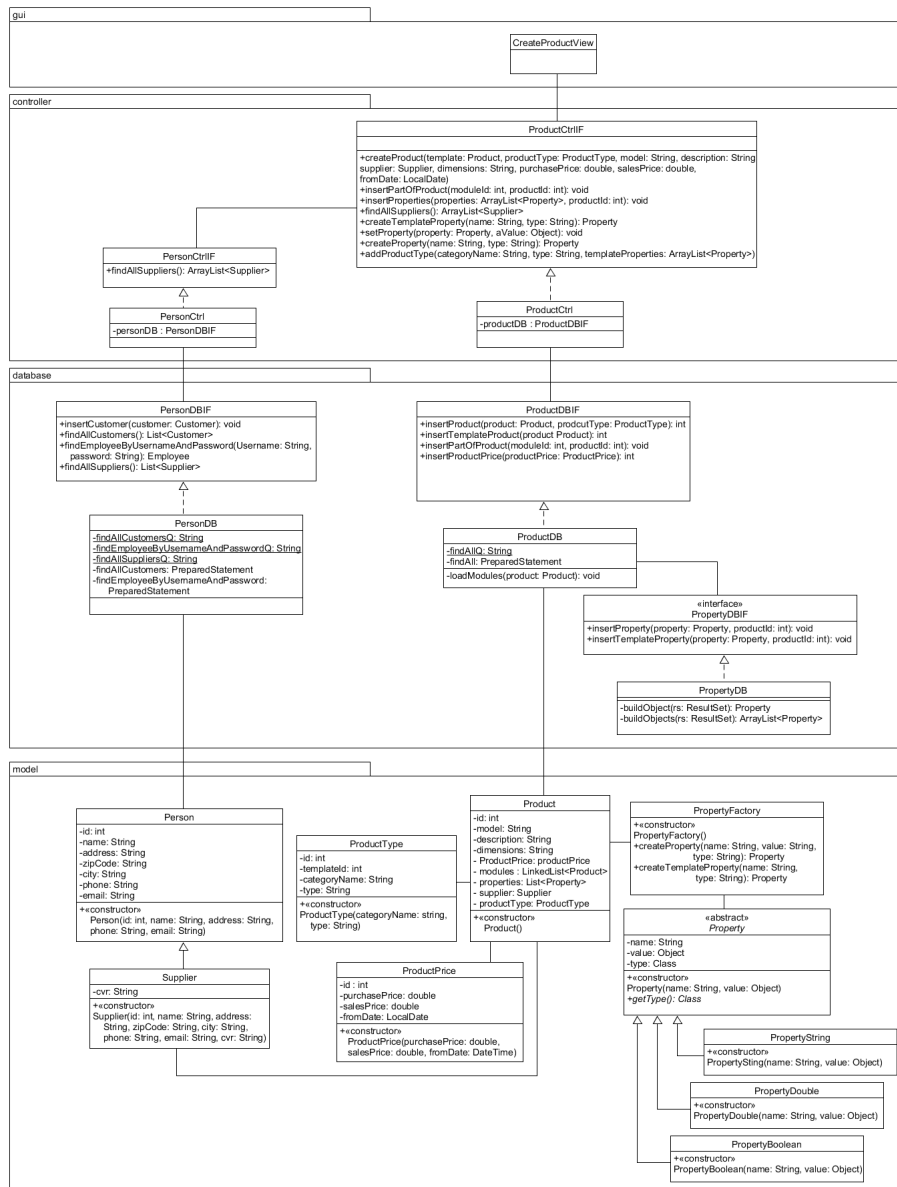
Ligesom tidligere modelleres kommunikationsdiagrammet ud fra system-sekvensdiagrammet og operationskontrakten. Kommunikationsdiagrammet til Opret produkt er illustreret på figur 4.12. Brugen af de tidligere omtalte designmønstre ses i trin 4. Brugeren vælger det ønskede *ProductType*-objekt *pt* ud fra en fremvist liste. I henhold til *Prototype Pattern* hentes det skabelon-objekt (*tempP*), som *pt* refererer til via sin *templateId* attribut, op fra databasen. I samme forbindelse kalder *ProductDB* i trin 4.4 *PropertyDB*-klassen som returnerer en liste af *Property*-objekter som har reference til *tempP*. Dette henviser til brug af *Property Pattern*.



Figur 4.12: Kommunikationsdiagram for use-casen opret produkt.

## Designklassediagram

På figur 4.13 ses et revideret udsnit af designklassediagrammet, som indeholder metoden der kræves for at lave use-casen opret produkt.



Figur 4.13: Designklassediagram passende til use-casen opret produkt

## 4.3 Iteration 3 - Opret produkttype

Efter iteration 2 revurderes iterationsplanen. Der besluttes at arbejde videre med den nye ProductType-klasse som er opstået i forbindelse med implementeringen af Type Pattern beskrevet i afsnit 4.2.1. Dette grundet ønsket om at komme mere i

dybden med alle aspekter af arkitekturen omkring håndtering af produkter.

### 4.3.1 Use-case revalidering

Da denne use-case ikke var med i inception fasens kravsspecifikation opstilles her en fully dressed.

Use-case	Opret produkttype	
Aktører	Lagermedarbejder	
Præbetingelse	Der er forbindelse til databasen.	
Postbetingelse	Produktet er blevet tilføjet til databasen med korrekte attributter. Man kan nu oprette produkter med den nye produkttype	
Frekvens	1 gang om måneden	
	Aktør	System
Flow of events	1. Bolighuset ønsker at føre en ny produkttype.	
	2. Medarbejderen vælger hvilken grundtype (hyldevare, specialdesign, modul) og udfylder kategorinavn	
	3. Medarbejderen vælger type på specialattribut og udfylder attributnavn.	
	4. Medarbejderen tilføjer specialattribut.	5. Systemet viser liste af tilføjede attributter
	<i>Trin 3-5 gentages til alle de ønskede attributter er tilføjet</i>	
	6. Medarbejder trykker på opret.	7. Systemet nulstiller skærm og tilføjer produkttypen til databasen
Alternative flow		3a. Der er ikke forbindelse til databasen 1. Systemet viser en fejlmeddelelse

Supplerende krav	<ol style="list-style-type: none"> <li>1. Brugeren skal fremvises mulige grundtype- og attributtypevalg</li> <li>2. Brugeren skal kunne for slette en tilføjet attribut</li> <li>3. I tilfælde af fejl skal brugeren altid have en forklarende fejlbeskrivelse.</li> </ol>
------------------	--

**Tabel 4.3:** Fully dressed beskrivelse for use-casen opret produkttype.

### 4.3.2 Arkitektur

I dette afsnit beskrives, hvilke tilføjelser til systemarkitekturen denne use-case har medført.

#### Factory pattern

Når der oprettes nye produkter, bruges der Property-objekter. Disse laves ved hjælp af det såkaldte factory pattern. Ved brug af factory pattern uddelegeres ansvaret for oprettelsen af et bestemt objekt til en enkelt klasse. Dette gøres for at gøre det enklere at oprette nye instanser af klasser, hvor der på forhånd ikke vides hvilken bestemt klasse der skal benyttes.

Måden hvorpå factory pattern implementeres, er ved at have en abstrakt klasse eller interface, som nogle andre klasser nedarver fra. Interfacet eller den abstrakte klasse indeholder metoder, som alle specialiseringsklasserne nedarver. Implementationen af factory pattern er vist på listing 4.5.

```

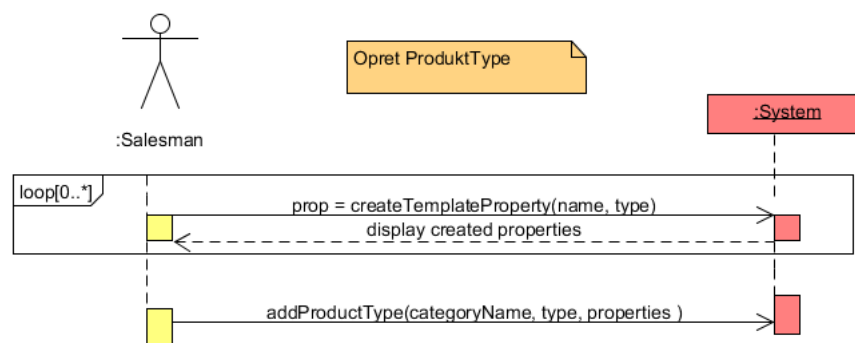
1 public Property createProperty(String name, String value,
   String type) {
2     switch (type) {
3         case "String": return new PropertyString(name, value);
4         case "Double": return new PropertyDouble(name,
   Double.parseDouble(value));
5         case "Boolean": return new PropertyBoolean(name,
   Boolean.valueOf(value));
6     }
7     return null;
8 }

```

---

**Listing 4.5:** Implementation af Property pattern.**System-sekvensdiagram**

I dette afsnit vises og beskrives system-sekvensdiagrammet der er udarbejdet i forbindelse med opret produkttype use-casen.



**Figur 4.14:** Systemsekvens-diagram for opret produkttype

Som det fremgår af systemsekvens-diagrammet på figur 4.14 er interaktionen mellem bruger og system begrænset i denne use-case. Brugeren forsyner systemet med input i form af kategori-navn og grundtype på den ønskede produkttype. Ønskes der tilknyttet specialattributter til den pågældende produkttype, gives der input til disses navn og type.

**Operationskontrakter**

Operationskontrakten på tabel 4.4 giver indblik i klasserne og instanserne forbundet med operationen `addProductType(...)`. Her ses der, hvordan de førømtalte designmønstre udspiller deres rolle i forbindelse med use-casen. I forbindelse med oprettelsen af et nyt `ProductType`-objekt oprettes automatisk et nyt objekt af typen `Product` som vil være skabelon. De tilføjede `Property`-objekter associeres med skabelon-objektet og `ProductType` samt `Product` associeres med hinanden.

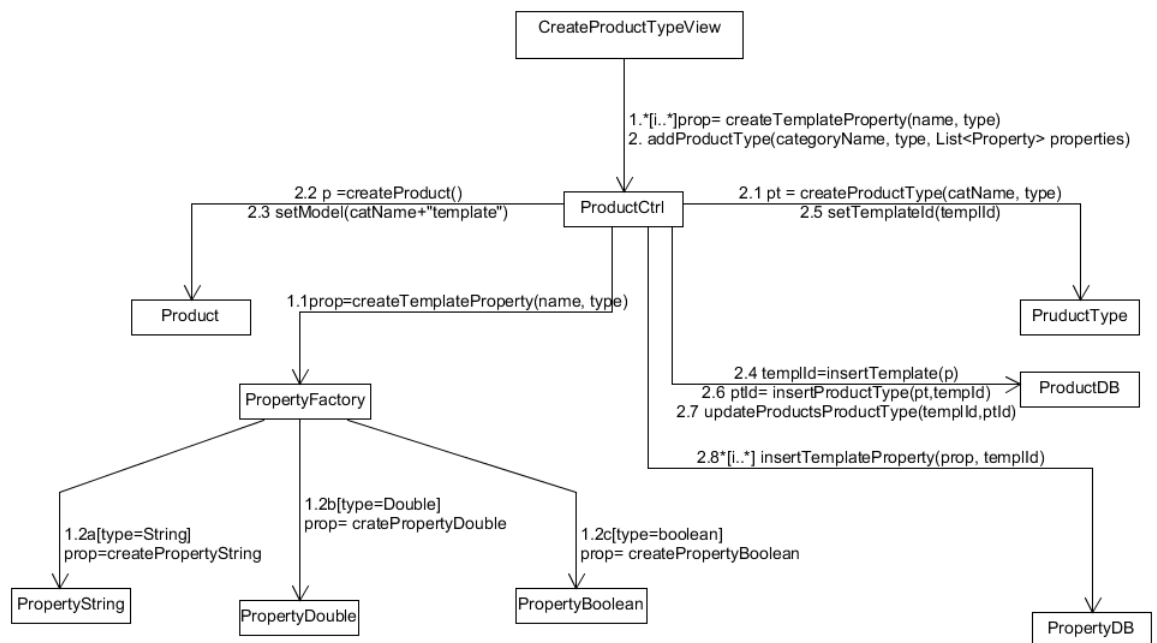


<b>Operation: addProductType(categoryName, type, List&lt;Property&gt;properties)</b>
Use-case: Opret produkttype <b>Præbetingelse:</b> - Nul til mange instanser <i>prop</i> af typen Property er blevet oprettet. <b>Postbetingelse:</b> - En instans <i>pt</i> af typen ProduktType blev oprettet. - <i>pt.categoryName</i> , <i>pt.type</i> blev tilskrevet værdier. - En instans <i>tempP</i> af typen Produkt blev oprettet og associeret til <i>pt</i> . - <i>tempP.model</i> blev tilskrevet værdien " <i>categoryName+template</i> ". - <i>pt.templateId</i> blev sat til <i>tempP</i> . - <i>prop</i> blev associeret til <i>tempP</i>

Tabel 4.4: Operationskontrakt af use-casen Opret produkttype

## Kommunikationsdiagram

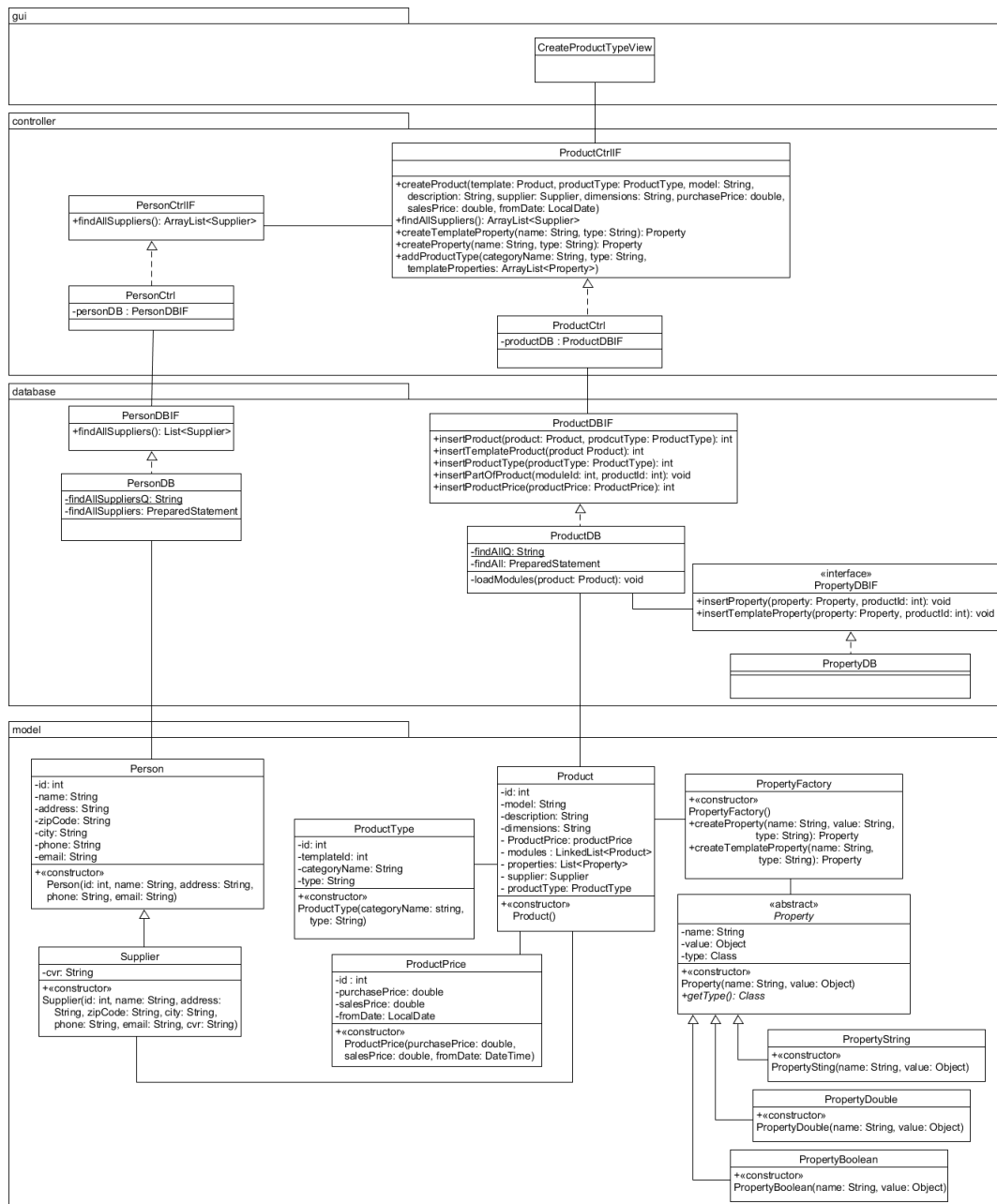
På kommunikationsdiagrammet, som er vist på figur 4.15, ses brugen af Factory Pattern i trin 1.1 og 1.2. GUI'en sender to String-værdier i form af name og type til ProductCtrl, som kalder PropertyFactory som ud fra type-attributten administrerer at skabe en instans af Property med den rette dynamiske type. Da både *p* og *pt* i kraft af deres gensidige association har brug for hinandens id-attribut for at skabe referencer i databasen og id'ene først bliver genereret ved indsættelse, kræver denne proces tre trin. Først indsættes *p* som returnerer det autogenererede id til ProductCtrl (trin 2.5). Dernæst indsættes *pt* med sin reference til *p* (trin 2.7). Med det returnerede id fra trin 2.6 bliver det gemte productskabelon-objekts reference til det ny-oprettede ProductType-objekt opdateret i trin 2.8.



**Figur 4.15:** Kommunikationsdiagram for use-casen Opret Produkttype.

## Designklassediagram

På figur 4.16 ses et revideret udsnit af designklassediagrammet som indeholder de aktuelle metoder der bruges af use-casen Opret produkttype.



Figur 4.16: Revideret designklassediagram til use-casen Opret produkttype.

### 4.3.3 Delkonklusion

Med de tre iterationer beskrevet og designet ved hjælp af system-sekvensdiagrammer, kommunikationsdiagrammer og designklassediagrammer vil der i næste afsnit beskrives noget af den kode, som er central for systemet.

## 4.4 Kode

I dette afsnit vil der blive gennemgået kodeeksempler og givet en forklaring på hvordan de er implementeret.

### Singleton

Til udvikling af systemet er Singleton mønstret benyttet. Dette sikrer at:

- Der kun oprettes ét adgangspunkt til databasen.
- Der kun oprettes én instans af konfigurations-klasser.
- Der altid er adgang til instansierede klasser.

Implementation af Singleton mønstret gøres ved, at implementere en statisk metode, som ses i metoden `getInstance()`, som er vist på listing 4.6.

```
1 public static DBConnection getInstance() {  
2     if (dbConnection == null) {  
3         dbConnection = new DBConnection();  
4     }  
5     return dbConnection;  
6 }
```

**Listing 4.6:** Implementation af `getInstance`-metoden.

Måden, hvorpå det sikres at der kun er en instans af objektet, er at `dbConnection` er statisk og derfor er den samme i hele delsystemet.

### SalesOrder

`SalesOrder`-klassen står for at oprette `SalesOrder`-objekter og holde på den information, som bliver tilknyttet til denne. Dette ses ved klassens attributter som er vist på listing 4.7.

```
1 public class SalesOrder {
2     private int id;
3     private LocalDate datePlaced;
4     private boolean paid;
5     private boolean orderSent;
6     private ArrayList<SalesOrderLine> salesOrderLines;
7     private OrderCondition orderCondition;
8     private double totalPrice;
9     private Customer customer;
10    private Employee employee;
```

**Listing 4.7:** Fields og constructor i SalesOrder-klassen.

Klassen har en lokal liste i form af en ArrayList kaldet salesOrderLines. Denne ArrayList består af SalesOrderLine-objekter, hvor Product'er og antallet af Product'er er tilknyttet. Desuden har en SalesOrderLine en Discount tilknyttet sig, som står beskrevet i afsnit 4.1.2. ArrayListen kan bestå af nul til mange SalesOrderLines. Siden en SalesOrderline har tilknyttet et Products salgspris afhængig af, om der er givet rabat på givne SalesOrderLine, benyttes metoden setTotalPrice() til at beregne den totale pris for en SalesOrder.

```
1 public void getTotalPrice() {
2     this.totalPrice = salesOrderLines.stream()
3         .mapToDouble(x -> x.getSubTotal())
4         .sum();
5 }
```

**Listing 4.8:** Metoden setTotalPrice.

Metoden getTotalPrice(), som er vist på listing 4.8, tager ArrayList'en salesOrderLine og udfører en stream-operation kaldet mapToDouble() og sum(). Dette gøres for at iterere gennem listen og lave en sum af alle Products subTotal priser. setTotalPrice() har en tidskompleksitet på  $O(n)$ , da metoden itererer gennem alle objekter i listen én gang.

#### 4.4.1 SalesOrderDB

SalesOrderDB er den databaseklasse som tilskriver SalesOrder til databasen. Tilskrivning til databasen foregår med metoden addSalesOrderToDatabase(), som er vist på listing 4.9.

```
1 @Override
```

```
2 public void addSalesOrderToDatabase(SalesOrder salesOrder)
   throws SQLException, PhoneAlreadyExistException,
   ZipAlreadyExistException, InsertFailedException {
3     insert.setDate(1,
4     java.sql.Date.valueOf(salesOrder.getDatePlaced()));
5     insert.setBoolean(2, salesOrder.isPaid());
6     insert.setBoolean(3, salesOrder.isOrderSent());
7     insert.setInt(4, salesOrder.getEmployee().getId());
8
9     if (new
10    PersonDB().findCustomerById(salesOrder.getCustomer().getId())
11    != null) {
12         insert.setInt(5, salesOrder.getCustomer().getId());
13     } else {
14         int id = new
15         PersonDB().insertCustomer(salesOrder.getCustomer());
16         insert.setInt(5, id);
17     }
18     int conditionID = new
19     OrderConditionDB().insertOrderCondition(salesOrder.
20     getOrderCondition());
21     insert.setInt(6, conditionID);
22     int id =
23     DBConnection.getInstance().executeInsertWithIdentity(insert);
24     //Insert SalesOrderLines:
25     ArrayList<SalesOrderLine> salesOrderLines =
26     salesOrder.getSalesOrderLines();
27     for(SalesOrderLine sol : salesOrderLines){
28         new SalesOrderLineDB().insertSalesOrderLine(id, sol);
29     }
30 }
```

**Listing 4.9:** Metoden addSalesOrderToDatabase.

Metodens første check er på linje 8, hvor der tjekkes efter om kunden eksisterer i databasen. Hvis det ikke er tilfældet, tilføjes kunde-objektet til databasen. Derefter indsættes en OrderCondition i databasen, hvor det generede id returneres. Efter dette indsættes selve ordren med de relevante attributter. Til sidst indsættes de SalesOrderLines-objekter som er oprettet sammen med ordren.

### 4.4.2 Samtidigshed

I dette underafsnit vises et kodeeksempel, i form af en løsning på et samtidigshed-problem, som kan opstå ved opdatering af en bestemt række i databasen.

Løsningen er implementeret ved at benytte SQL-metoder. Metoderne er `commit()`, `setAutoCommit(boolean)` og `rollback()`. Disse bruges til at sikre, at ændringer ikke kan ske i samme datasæt, så længe en bruger har taget dataet i brug. Dette resulterer i, at når en ansat henter en kunde op fra databasen, så kan en anden ansat ikke ændre i den samme kundes data, da denne kunde er låst af den første ansatte. Dette ses i metoden som er vist på listing 4.10.

```
1      @Override
2      public void updateCustomer(String name, String address,
3          String zipCode, String city, String currPhone,
4          String newPhone, String email, String person_type,
5          String type) throws NotImplementedException, SQLException {
6          try {
7              DBConnection.startTransaction();
8              personDB.updateCustomerByPhone(name, address, zipCode,
9              city, currPhone, newPhone, email, person_type, type);
10             DBConnection.commitTransaction();
11         } catch (Exception e) {
12             DBConnection.rollbackTransaction();
13         }
14     }
```

Listing 4.10: Implementation af samtidigshed

### Connection

I dette afsnit vises og beskrives metoden `startConnectionStatus()` fra klassen `CreateSalesOrderView`. Metoden er vist på listing 4.11. Denne metode instantierer en tråd som løbende tjekker efter forbindelse til databasen og sætter en `JLabel` som virker som statusindikator for brugeren. I et infinite while-loop køres et condition-tjek iform af en `if-else` statement. If-blokken kalder `DBConnection.getConnection().isValid()` som returnerer en boolean som indikerer om der er forbindelse til databasen. Hvis der er forbindelse sættes `JLabel`en `lblOnline` til at skrive "online" og ændre skrifttype til grøn. Hvis der ikke er forbindelse køres else-blokken som sætter `lblOnline` til at skrive "offline" og ændre skrifttype til rød. I slutningen af metoden sættes tråden til at sove i et sekund, hvorefter loopet køres igen.

```
1  protected static void startConnectionStatus() {
2      Thread t1 = new Thread(() -> {
3          while (true) {
4              try {
5                  if
6                      (DBConnection.getInstance().getConnection().isValid(0)) {
7                      setConnectionLabel("Online");
8                      lblOnline.setForeground(new Color(0, 128, 0));
9                  } else {
10                     setConnectionLabel("Offline");
11                     lblOnline.setForeground(new Color(255, 0, 0));
12                     DBConnection.getInstance();
13                 }
14                 Thread.sleep(1000);
15             } catch (Exception e) {
16                 e.printStackTrace();
17             }
18         });
19     t1.start();
20 }
```

Listing 4.11: Connection metode

#### 4.4.3 Delkonklusion

De fremviste metoder er essentielle for udførelsen af diverse use-cases og skrivning og læsning af data fra databasen, som opfylder det der er nødvendigt for at have et funktionelt program. I næste afsnit vil der udvikles tests for at sikre at koden bliver ved med at fungere.

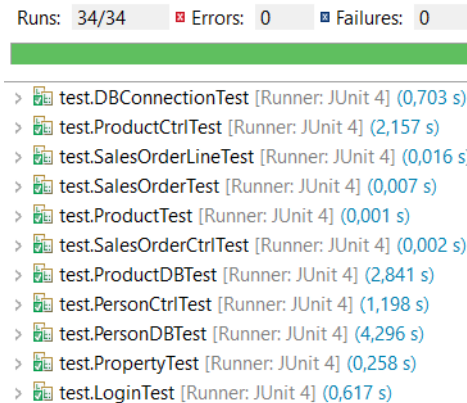
### 4.5 Test

I dette afsnit vil der blive opstillet test-cases. Disse er både system- integrations- og unittests.



### 4.5.1 Unittests - Whitebox

Men systemet blev udviklet har det været praksis løbende at teste implementeret kode. Dette er blevet gjort ved udføre små whitebox unit-tests på de implementerede metoder. Til dette formål er der blevet anvendt test-frameworket JUnit.



Figur 4.17: Unittests kørt ved hjælp af frameworket JUnit.

Som det ses på figur 4.17 er der blevet udarbejdet 34 tests i alt som er fordelt på model-, controller- og databaselagene. I det følgende vises et par eksempler på udviklede tests.

#### SalesOrderTest

I denne test-case tjekkes om aritmetikken omkring udregning af den totale pris bliver udregnet korrekt. Testen hedder `getTotalPriceTest()` og er vist på listing 4.12.

```

1  @Test
2  public void getTotalPriceTest() {
3      try {
4          assertEquals(11000, o1.getTotalPrice(), 0);
5          assertEquals(13900, o2.getTotalPrice(), 0);
6          assertEquals(10900, o3.getTotalPrice(), 0);
7          o2.getSalesOrderLines().get(0).setDiscount("1000");
8          assertEquals(12900, o2.getTotalPrice(), 0);
9          o3.getSalesOrderLines().get(1).setDiscount("10%");
10         assertEquals(9900, o3.getTotalPrice(), 0);
11     } catch (Exception e) {
12         e.getMessage();
13     }

```

```
14 }
```

**Listing 4.12:** getTotalPriceTest metoden.

Variablerne o1, o2 og o3 er alle af typen SalesOrder, og beregningerne for hvad de forskellige salgsordre er blevet manuelt beregnet for at kunne teste, om metoden er implementeret korrekt.

```

1      @Test
2      public void insertCustomerInDatabase() throws SQLException {
3          int beforeInsert = personDB.getRowCountFromPersonTable();
4          try {
5              personDB.insertCustomer(customer);
6          } catch (SQLException | PhoneAlreadyExistException |
7              ZipAlreadyExistException | InsertFailedException e) {
8              e.printStackTrace();
9          }
10         int afterInsert = personDB.getRowCountFromPersonTable();
11         assertEquals(beforeInsert, afterInsert - 1);
12         Customer tempCustomer =
13             personDB.findCustomerByPhone(customer.getPhone());
14         assertEquals(customer.getAddress(),
15             tempCustomer.getAddress());
16         assertEquals(customer.getZipCode(),
17             tempCustomer.getZipCode());
18         assertEquals(customer.getEmail(), tempCustomer.getEmail());
19         assertEquals(customer.getPhone(), tempCustomer.getPhone());
20         personDB.removeCustomerByPhone(customer.getPhone());
21     }

```

**Listing 4.13:** Testcasen insertCustomerInDatabase

I test-casen, som er vist på listing 4.13, tilføjes en Customer til databasen, hvorefter der ses om antallet af rækker der er i person-tabellen er blevet en større. Efterfølgende findes objektet frem igen, og der tjekkes om den fundne Customers fields matcher de værdier, som forventes blev skrevet til databasen.

I test-casen, som er vist på listing 4.14, findes først antallet af SalesOrder i databasen. Derefter kaldes metoden closeSalesOrder() som indsætter et SalesOrder-objekt i databasen. Derefter sammenlignes antallet fra før med det aktuelle. Efterfølgende tjekkes der, at SalesOrder er blevet sat til null.

```

1      @Test
2      public void closeOrderTest() {
3          try{

```

```
4     salesOrderCtrl.createSalesOrder();
5     int before = salesOrderDB.findAll().size();
6     salesOrderCtrl.closeSalesOrder();
7     assertEquals(before, salesOrderDB.findAll().size());
8     assertNull(salesOrderCtrl.getOrder());
9 } catch (Exception e) {
10     e.getMessage();
11 }
12 }
```

**Listing 4.14:** Testcasen closeSalesOrderTest

### 4.5.2 Systemtest

Nedenfor ses opstillede systemtest-cases for use-casen opret salgsordre. Test-casene er opbygget om forskellige scenarier som tager udgangspunkt i forskellige flows, som er beskrevet i fully dressed use-case beskrivelserne. Disse tests er udformet som black-box tests, hvor der udelukkende ses på input og forventet output. På figur 4.18 ses et uddrag af test-matrixen for systemtests vedrørende opret salgsordre. Her ses bort fra konkret data. Test-casenes forløb samt input og output beskrives abstrakt. Resten af disse test-cases kan findes i bilag H.

Test case id	Scenarie/ betingelse	Input				Forventet Resultat
		Vareid	Kunde	antal	discount	
SalesOrder _01	Scenarie 1: Sussesfuld ordre	V	V	V	V	<ul style="list-style-type: none"> <li>• Varebeskrivelse, varekategori og pris svarer til valgte vare.</li> <li>• Telefonnr, adresse, email, svarer til søgte kundenavn.</li> <li>• Deltotal på ordrelinje passer med varens pris gange antal minus den angivne discountsats.</li> <li>• Total stemmer med summen af subtotaler.</li> </ul>
SalesOrder _02	Scenarie 2: Ulovligt antal	V	N/A	I	N/A	Hvis antal er et negativ tal sættes antal til 0 og der gives en fejlmedling til bruger.
SalesOrder _03	Scenarie 3: Ulovligt discount	V	N/A	V	I	Hvis discountsatsen underskrider 0% eller overskrider 100% sættes discount tilbage til 0 og bruger modtager en fejlmedling.

**Figur 4.18:** Systemtest-cases for opret salgsordre. Uddrag af scenarier

På figur 4.19 ses de samme test-cases med tilføjede dataværdier for valid og invalid input (v og i) samt outputs. Resultaterne med aktuelle værdier kan også findes i bilag H.

Test case id	Scenarie/ betingelse	Input				Forventet Resultat
		Vareid	Kunde	antal	discount	
SalesOrder _01	Scenarie 1: Sussesfuld ordre	Atlanta	Bodil Pallesen Eller 5566778 8	2	7%	Varebeskrivelse = "Sofa opstilling 09", Kategori = "Sofa", Deltotal = 1840kr, Salgspris = 1000 kr Adresse = "sidevej 5", Postnr =9400, By= "Noerresundby" Email = "bodil@pallesen.dk Total = 1840kr
SalesOrder _02	Scenarie 2: Ulovligt antal	Atlanta	N/A	-2	N/A	Antal = 0
SalesOrder _03	Scenarie 3: Ulovligt discount	Atlanta	N/A	3	-5% 101%	Discount = 0%
SalesOrder _04	Scenarie 4: Ugyldigt kundenavn eller -telefonnr.	N/A	Bodil Pallesen Eller 5566778 9	N/A	N/A	Popup vidue med teksten "Ugyldigt kundenavn eller telefonnr" Navn, telefon, adresse, postnr, by, email = ""

**Figur 4.19:** Systemtest-cases for opret salgsordre med tilføjede dataværdier. Uddrag af scenarier

### 4.5.3 Integrationstests

Med udgangspunkt i kommunikationsdiagrammerne identificeres kritiske metoder med ækvivalensklasser, hvortil der er blevet opstillet black-box integrations-tests. Der findes ikke mange ækvivalensklasser i forbindelse med use-casen opret salgsordre. På figur 4.20 ses de udformede test-cases for metoden setDiscount som indeholder flest regler og ækvivalensklasser. Der fokuseres på at teste i grænseområderne for de ækvivalente klasser.

**setDiscount(String amount):**

**Involverede klasser:** *SalesOrderLine*, *DiscountByPercent*, *DiscountByAmount*

**Regler:**

- Discount kan angives i hele tal, decimaltal og i procent.
- Systemet skal selv finde ud af, hvilken DiscountIF instans der skal benyttes.
- Discount må ikke være under 0kr, 0% eller overstige deltotal, 100%.

**Ækvivalensklasser for amount**

I procent:

Amount < 0% : Invalid

Amount > 100% :Invalid

100% >= Amount >= 0%: Valid

I tal:

amount < 0 : Invalid

amount > deltotal: Invalid

deltotal >= amount >= 0 : Valid

Metode: setDiscount(String amount)			
Test case	Deltotal før	Input	Forventet resultat
O1	3000	750, 750.0 25%, %25	discount = 750 deltotal = 2250
O2	3000	-1, 3001, -0.5%, 101%	discount = 0 deltotal = 3000 IllegalDiscountException
O3	3000	0, 0%, 0	discount = 0 deltotal = 3000
O4	3000	3000, 100%	discount = 3000 deltotal = 0
O5	3000	195.5	discount = 195.5 deltotal = 0
O6	3000	Ab3c#	discount = 0 deltotal = 3000 NumberFormatException

**Figur 4.20:** Integrationstest-cases for metoden setDiscount.

På listing 4.15 ses en automatisering af fornævnte test-cases ved hjælp JUnit. Objektet *SalesOrderLine* objektet *sol* er sat op i test-setup til at have en *subTotal* på 3000 kr.

```

1  @Test
2  public void setDiscountTest(){
3      String[] valids = {"750", "25%", "%25", "750.0"}; String[]
      invalids = {"-1", "3001", "-0.5%", "101%", "Ab3c#"};
4      for(String s : valids){
5          try {

```

```
6         sol1.setDiscount(s);
7         assertTrue(sol1.getDiscount() == 750);
8         assertTrue(sol1.getSubTotal() == 2250);
9     } catch (Exception e) {
10         fail();
11     }}
12     for (int i = 0; i < invalids.length; i++) {
13         String s = invalids[i];
14         try {
15             sol1.setDiscount(s);
16             fail();
17         } catch (IllegalDiscountAmountException e) {
18             assertTrue(sol1.getDiscount() == 0);
19             assertTrue(sol1.getSubTotal() == 3000);
20             if(i==0 || i==1){assertEquals(e.getMessage(), ''Rabat
kan ikke vaere under 0 eller over originalpris'');}
21             else{assertEquals(e.getMessage(), ''Rabat kan ikke
vaere under 0% eller over 100%'');}
22         }
23         catch(NumberFormatException e) {
24             assertTrue(i == 4);
25             assertTrue(sol1.getDiscount() == 0);
26             assertTrue(sol1.getSubTotal() == 3000);
27         }
28     }
29     sol1.setDiscount("0%");
30     assertTrue(sol1.getDiscount() == 0);
31     assertTrue(sol1.getSubTotal() == 3000);
32     sol1.setDiscount("0");
33     assertTrue(sol1.getDiscount() == 0);
34     assertTrue(sol1.getSubTotal() == 3000);
35     sol1.setDiscount("100%");
36     assertTrue(sol1.getDiscount() == 3000);
37     assertTrue(sol1.getSubTotal() == 0);
38     sol1.setDiscount("3000");
39     assertTrue(sol1.getDiscount() == 3000);
40     assertTrue(sol1.getSubTotal() == 0);
41 }
42 }
```

Listing 4.15: Testcasen setDiscountTest

#### 4.5.4 Delkonklusion

Ved udarbejdelse af tests er det blevet bekræftet, at implementationen for de valgte use-cases umiddelbart fungerer som forventet. Systemet er derfor vurderet klar til at blive testet hos kunden.

### 4.6 Usability-evaluering

I dette afsnit beskrives en usability-evaluering, som er foretaget hos Brovst Bolig-hus. Formålet med at foretage en usability-evaluering er at få forståelse for, hvorvidt brugergrænsefladen kan benyttes, og forbedre designet af systemet.

Til denne test blev de scenarier der findes i bilag G benyttet.

Testpersonen er en af de personer som skal benytte systemet, når det er færdigudviklet. Det er af denne årsag vigtigt at han finder systemet intuitivt at benytte.

I den første opgave, hvor det første trin er at logge ind, kom testpersonen til at lave et mellemrum ved brugernavnet. Dette resulterede i, at selvom den rigtige kode blev indtastet, skrev systemet, at der var enten forkert brugernavn eller kodeord. Da fejlen blev fundet, og han var logget ind, var han hurtig til at komme ind i "opret ordre" vinduet, hvorfra resten af opgaven blev løst uden problemer.

Opgave to blev løst uden problemer. Dette kan skyldes, at testpersonen allerede havde opdaget, at der er to forskellige kategorier af produkter.

Testpersonen nævnte ved den tredje opgave, at han allerede havde opdaget, at der er to muligheder for at oprette kunder. Opgaven blev dog løst på den tiltænkte måde.

Ved fjerde opgave fungerede systemet ikke hensigtsmæssigt, og det var derfor ikke muligt for testpersonen at udføre opgaven. Af denne årsag blev der identificeret en fejl i systemet.

Den femte opgave blev udført uden problemer.

Både under og efter testen kom testpersonen med mange forskellige bemærkninger om, hvad der fungerede, og hvad der fungerede knap så godt. Problemerne og løsningerne på disse er beskrevet på nedenstående tabel (tabel 4.5).

Problem	Mulig løsning	Problem løst
---------	---------------	--------------



Fra login-skærmen kan man ikke trykke på enter for at logge ind	Brug en key-listener til at tjekke, om der bliver trykket på enter	Ja
De forskellige menuer i hoved-skærmen har meget lille tekst. Desuden kan det være fint med mere genkendelighed og ikoner i stedet for tekst	Lav teksten større og brug samtidig ikoner til at illustrere, hvad der håndteres	Nej
Programmet nulstiller ikke de felter som udfyldes, når en salgsordre oprettes	Lav en metode som kan nulstille alle felter, der ændres i systemet	Nej
Der er ingen alfabetisk rækkefølge i navne på varer	Sortér produkterne automatisk efter navn	Ja
Der mangler mulighed for at kunne tilføje ekstra kommentarer til leverandøren, som eksempelvis farvevalg på sofa, eller at der skal andre ben på	Tilføj et kommentarfelt	Nej
Postnummer skal indtastes før bynavn	Flyt postnummer op over by	Ja
Det vil være fedt, hvis by automatisk hentes ud fra postnummer	Tilføj postnumre og byer til databasen og hent postnummeret op derfra – eller hent postnummer og by fra internettet	Nej
Det er ikke muligt at indtaste område i byen, så man kan have svært ved at finde adressen, når varen skal leveres	Tilføj en område-attribut	Nej
Der kan ikke tilføjes EAN og CVR-nummer til en erhvervskunde	Tilføj EAN og CVR-attributter til erhvervskunder	Nej
Der skal gerne være mulighed for at generere en faktura til kunden samtidig med salgsordren oprettes	Lav funktionalitet til at generere en faktura	Nej
opret salgsordre vil fint kunne være hovedskærm, da det er denne som bruges oftest	Gør opret salgsordre vinduet til hovedskærm og lav de andre menuer til en palette med menupunkter, hvorfra brugeren kan åbne andre vinduer	Nej

Man skal kunne se, hvor stor en del af beløbet der er moms på en salgsordre	Tilføj moms-satser i systemet samt en momslinje på salgsordren, hvor der står hvor meget moms der er i alt	Nej
Man skal kunne ændre på alle attributterne i salgsordrelinjerne, sådan det er nemmere at ændre f.eks. antal varer	Lav alle felterne redigerbare	Nej
Det skal være muligt at tilføje en samlet kommentar / rabat	Gør det muligt at lave ekstra "salgsordrelinjer", som ikke er salgsordrelinjer	Nej
Når salgsordren skal sendes til leverandøren, skal salgsordren være over et bestemt beløb	Lav et tjek som sikrer, at beløbet er stort nok til at varerne bliver leveret fragtfrit	Nej
Hvis der er gået lang tid siden salgsordren er blevet oprettet, men salgsordren endnu ikke er sendt til leverandøren kan man glemme at der er en bestilling til den pågældende leverandør	Tjek om der er gået mere end et bestemt antal dage, siden salgsordren blev oprettet	Nej

**Tabel 4.5:** Problemer identificeret ved usability-evalueringen med tilhørende løsninger.

Derudover blev der nævnt at det er en god idé at alle produkter har et billede tilknyttet, sådan at det også er muligt at se på billedet hvilket produkter der er valgt.

## 5 | Diskussion

I dette kapitel diskuteres, hvad der kunne være gjort anderledes i løbet af projektperioden for at lette eller forbedre udviklingsprocessen i forhold til projektet.

### 5.1 Virksomhed

Det kan diskuteres hvorvidt cost-benefit er præcist, omkring dens nøjagtighed af den reelle return of investment, da beløbet som er givet i cost-benefit, er højt, siden dette beløb er udregnet efter *best-case scenario*. Derfor kan tallene være misvisende, i forhold til hvad det reelle tal ville have været.

Dette kan argumenteres, da vi som udviklere kan fordreje disse tal, således tallene bliver det som er bedst for os, og ikke nødvendigvis for virksomheden.

### 5.2 Systemet

I dette afsnit beskrives de overvejelser og valg der har været i forbindelse med udvikling af systemet. Der diskuteres desuden hvorvidt succeskriterierne er opfyldt.

#### 5.2.1 Valg af use-cases

I forbindelse med use-case prioritering i afsnit 3.7 og valget af use-cases til bearbejdelse i iterationer, skal valgene ses ud fra dette projekts kontekst. Med dette menes der, at da dette projekt repræsenterer en læringssituation er valget af use-cases også truffet ud fra deres læringsværdi. For gruppen repræsenterede problematikken omkring håndtering af de mange forskelligartede vare med unikke attributter en spændende udfordring og en mulighed for at tillære nye designmønstre som kan

anvendes til design af fremtidig softwarearkitektur

### 5.2.2 Funktioner som kunne være implementeret anderledes

På nuværende tidspunkt bliver *Password* gemt som en *String*. Dette er ikke den bedste løsning, eftersom at denne ikke som sådan er gemt for brugere der kan få adgang til databasen. Der er dog ikke en funktion i grænsefladen som gør, at password kan findes. *Password* burde hashes, så det ikke kan aflæses i databasen på en nem måde. Dette betyder at password på nuværende tidspunkt gemmes på en meget dårlig måde i databasen, og såfremt ondtindene individer skulle få adgang, ville det være muligt at finde alle passwords og usernames.

### 5.2.3 Funktioner der endnu ikke er implementeret

I dette afsnit beskrives funktioner der endnu ikke er implementeret og hvordan de kunne implementeres.

#### Flere telefonnumre tilknyttet en enkelt kunde

Der har været tale om, at det skulle være muligt, at en enkelt kunde skal kunne have mere end et telefonnummer. Denne funktionalitet er implementeret i databasen, men endnu ikke i selve systemet. En måde hvorpå funktionaliteten kan implementeres i systemet, er ved at bruge en *ArrayList*. Hvis man gerne vil kende forskel på de forskellige telefonnumre, er det nødvendigt at tilføje en kolonne mere til databasen som indeholder, hvilken type telefonnummer det er. Derudover skal der i systemet bruges en datastruktur, som kan indeholde både nummeret og typen - eksempelvis et *HashMap*. I systemet er databasen allerede forberedt til at kunne indeholde flere telefonnumre til den samme kunde. Det er dog ikke muligt at kende forskel på typen, og der skal derfor tilføjes en række mere i tabellen til at indikere typen af telefonnummer.

#### Automatisk e-mail til leverandør

Ifølge use-casen opret salgsordre vil der ved afslutning blive afsendt en e-mail til leverandøren med salgsordren. Dette er ikke implementeret. Det er ønskværdigt, at der ved afslutningen af en salgsordre samles flere ordrer inden en e-mail sendes. Hvis leverandøren ikke på forhånd samler flere salgsordrer og afsender mere end

én af gangen, kan dette spare Brovst Bolighus leveringsomkostninger. En bestilling til leverandøren bør ikke indeholde mere information end kundens navn som reference og de produkter som skal bestilles.

### Udskriv faktura

Brovst Bolighus ønsker at kunne printe faktura direkte fra opret salgsordre vinduet. Denne funktion er endnu ikke implementeret. Den ville fungere ved at sende en faktura over alle produkter som er valgt til en salgsordre og dertilhørende information. Disse ville blive sendt til kundens valgte e-mail og derefter blive udskrevet eller gemt i en database for at føre arkiv.

#### 5.2.4 Duplikering af informationer

Da ansatte ikke kan fungere som kunder i systemet er det nødvendigt. Dette er ikke umiddelbart et problem, da der i virksomheden er få ansatte. Dog kan det blive et problem, hvis virksomheden var meget større. En løsning på dette problem er at gøre, sådan ansattes informationer genbruges, når de handler hos virksomheden.

#### 5.2.5 Opfyldelse af succeskriterierne

I indledningen blev der defineret nogle succeskriterier. I det følgende vil der diskuteres, hvorvidt de enkelte kriterier er blevet opfyldt.

1. Systemet er stabilt og indeholder ingen kritiske fejl samt pludselige nedbrud.
  - Det er så vidt muligt forsøgt at gøre programmet stabilt ved at lave fejlhåndtering. Derudover bør en opdatering i databasen også blive helt gennemført eller slet ikke, da der er gjort brug af *transactions*, som gør at alle ændringer som har noget med databasen at gøre, skal være gennemført 100 %, ellers rulles der tilbage til lige før transaktionen startede.
2. Systemet hjælper de ansatte med bestilling af varer.
  - Resultatet af usability-evalueringen viser, at systemet umiddelbart vil hjælpe de ansatte med at bestille varer, og derfor anses dette succeskriterie som opfyldt.
3. Systemet skal målbart nedsætte den tid det tager at udføre opgaver der tidligere blev udført på papir.
  - Dette er ikke blevet testet, og det er derfor ikke muligt at vurdere, hvorvidt succeskriteriet er blevet opfyldt. Det vurderes dog, at det vil tage

tid før systemet vil være hurtigere, da virksomheden har brugt den eksisterende metode i mange år. Systemet vil dog umiddelbart fra starten effektivisere ordrehåndtering og arkivering.

4. Systemet skal kunne fortælle, hvor en markedsføringskampagne rammer flest mulige kunder fra målgruppen.
  - Denne funktionalitet er endnu ikke implementeret.
5. Systemet skal til enhver tid kunne fortælle en medarbejder, om en vare er på lager eller hvornår den forventes at ankomme.
  - Denne funktionalitet er ej heller implementeret.

## 5.3 Proces

I denne sektion vil der blive diskuteret de forskellige processer gruppen har været i gennem ved udarbejdelse af dette projekt.

### 5.3.1 Fremgangsproces

Til udførsel og udvikling af rapport og det tilhørende program, SmartOrder, er en iterativ model blevet anvendt. Den anvendte model er Unified Process (UP), som er en plan-dreven processtilgang. For at benytte UP som udviklingsmodel, blev en UP faseplan lavet med estimater på datoer, hvor bestemte milepæle i udviklingen skulle opnås. Denne faseplan kan ses på tabel 5.1. Datoerne er fordelt på de forskellige faser, for at kunne overholde den tidsgrænse, som er fastsat af uddannelsen. Til de store milepæle blev mindre mål fastlagt, så der i fællesskab kan dannes et overblik over, hvad der mangler, og hvad der var udviklet på et givent tidspunkt, og på den måde er faseplanen midtpunktet for udviklingen i UP-modellen. Faseplanen viser antal iterationer som er blevet gennemgået i hver fase. Grundet tidsgrænsen var det ikke muligt fra starten at sætte antal på, men derimod arbejde iterativt med en use-case af gangen, og derefter vurdere, om der var tid til endnu en.

Faseplan	Analyse	Inception	Elaboration	Construction	Transition
Mål	Analyse af virksomhed - business case	Vision, afgrænsning og business case	Arkitekturen og arkitekturbærende funktionalitet designes, kodes og testes	use-cases designes, kodes og testes	Systemet testes i driftsmiljø
Antal iterationer	1	2	3	1	0
Færdig dato	Ultimo marts	10. april	15. maj	Ultimo maj	?
Artefakter	<ul style="list-style-type: none"> <li>• Nuværende situation</li> <li>• Strategianalyse</li> <li>• IT-strategi</li> <li>• Applikationer og information</li> <li>• Cost-benefit analyse</li> <li>• Plan og estimer</li> <li>• Brugerdeltagelse</li> </ul>	<ul style="list-style-type: none"> <li>• Vision</li> <li>• Mock up</li> <li>• Use-case</li> <li>• Domæne-model</li> <li>• Proof-of-concept</li> <li>• Risici</li> </ul>	<ul style="list-style-type: none"> <li>• SSD</li> <li>• UI design</li> <li>• Eksekverbare arkitektur/mønstre</li> <li>• Database-design</li> <li>• Use-case design</li> <li>• Implementering</li> <li>• Testcases/test</li> <li>• Løsninger på risici</li> <li>• Plan og budget</li> </ul>	<ul style="list-style-type: none"> <li>• Revisioner af use-cases</li> <li>• Use-case design</li> <li>• Implementering</li> <li>• Testcases/test</li> <li>• Dokumentation</li> <li>• Forberedelse af deployment</li> </ul>	<ul style="list-style-type: none"> <li>• Beta tests</li> <li>• Deployment</li> </ul>

Tabel 5.1: Faseplan lavet til at styre projektet.

Den første iteration var analyse-delen, hvori vi analyserede virksomhedens behov og potentielle muligheder for nye IT-tiltag. Her blev der bygget en businesscase, og hvor den nuværende situation blev analyseret og fundet frem til hvilken IT-strategi der skulle benyttes for et nyt system. Dette tog udgangspunkt i et interview med Brovst Bolighus, hvor det blev etableret, at der var potentiale i at udvikle tre forskellige systemer, nemlig lagerstyring, håndtering af salgsordre og et system til håndtering af markedsføring. Ved hjælp af en cost-benefit analyse samt plan og

estimer, blev det besluttet at ordre og markedsførings-delsystemerne havde størst forretningsværdi. På baggrund af kompleksitetskrav og forretningsværdi blev det valgt at arbejde videre med et ordre-håndteringssystem.

De næste to iterationer var i fasen inception, hvor der blev opstillet vision med systemet, og derved designet en domænemodel for systemet. For at være sikker på at systemet kunne laves, var det nødvendigt at minimere risici i forhold til udvikling af systemet, såfremt der skulle bruges kompetencer som endnu ikke var udviklet i forhold til gruppen. Vi benyttede os af proof-of-concept til at undersøge, at vi kunne overholde vores design af systemet. Dette var til vores property-klasse som kan ses på domænemodellen på figur 3.8, som allerede vakte tvivl i inception-fasen, da mønstret ikke var blevet brugt førhen. Dette gik godt og mindskede derfor risici i forhold til yderligere udvikling, da vi vidste, at det kunne lade sig gøre med de tilegnede kompetencer.

De næste tre iterationer var i elaboration-fasen, hvor de konkrete use-cases og diagrammer blev udviklet. Den første iteration var use-casen opret salgsordre, herefter opret produkt og den sidste use-case opret produkttype. Her blev fundamentet og den mest arkitekturbærende funktionalitet designet og implementeret ved hjælp af designklassediagrammet som på baggrund af hver use-case blev opdateret. Hver use-case gennemgik sin egen iteration og blev udviklet og implementeret ved hjælp af Trello taskboard, hvori iterationer blev oprettet som forskellige opgaver, så det kunne ses, hvor langt i iterationen projektgruppen var kommet. Her blev taskboardet også brugt til at se, hvem der var på hvilke opgaver for at projektstyre, i forhold til hvad man skulle i gang med som det næste. I disse iterationer blev der også udviklet test-cases og tests i forhold til nødvendigt funktionalitet i de forskellige use-cases.

Den sidste iteration er en iteration i construction-fasen. Her er der blevet brugt tid på at overveje designvalg i forhold til use-cases og revidering af use-cases. Det er også i denne iteration at en mindre fjerde use-case blev udviklet. Denne sidste use-case er opret kunde. Det sidste i forhold til implementering og test er blevet færdiggjort. Dokumentation til den færdige kode er også blevet lavet i den sidste iteration. Den sidste iteration er dog hvor man i processen kan se, at dette er et projekt med en deadline. Dette bærer iterationen præg af, da der er blevet brugt en del tid på at gennemgå alt som tidligere er produceret, ikke blot i forhold til kode, men også til selve rapporten.



### 5.3.2 Implementationsproces

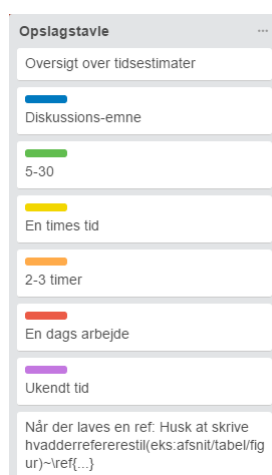
Som nævnt i afsnit 5.3.1 har størstedelen af projektarbejdet foregået ved hjælp af Unified Process, hvilket har gjort, at use-casen opret salgsordre har været nem at implementere, da de krævede metoder allerede var fremvist i de udarbejdede diagrammer. Der er dog i processen set tegn på agil projektudvikling, hvilket skyldes et til tider rodet taskboard, hvor det kunne ske, at der blev gået videre med noget nyt, selvom en forrig iteration ikke var færdig.

Til implementering af programmet er der blevet gjort brug af pair-programming. Dette er gjort for at mindske mængden af fejl der nemt kunne opdages, når en person havde overblik over diagrammet der blev kodet ud fra og en anden for selve programmeringen. Dette fremmer desuden læring, da individer i gruppen har forskellige erfaringer, og der sikres dermed, at alle får indflydelse på de forskellige dele af arbejdet.

### 5.3.3 Arbejdsproces

Udarbejdelse af rapporten har for det meste været uddelegering af afsnit eller stykker af tekst der skulle produceres. I de fleste tilfælde er disse blevet skrevet individuelt med senere gennemgang fra andre gruppemedlemmer for kvalitetssikring. Ved udarbejdelse af eksempelvis cost-benefit og diverse diagrammer blev der dannet små grupper. Da *domænemodellen*, *kommunikationsdiagrammerne* og *designklasse-diagrammet* skulle designes blev dette gjort i samarbejde med hele gruppen. Projektet er skrevet i redskabet ShareLatex, som er en realtids klient bygget på L<sup>A</sup>T<sub>E</sub>X, som har gjort skrive arbejdet en del lettere.

For at holde opgaver overskuelige er der brugt en fælles opslagstavle på Trello.com. På denne hjemmeside er det muligt at oprette og fordele opgaver. Vi har desuden brugt labels til at allokere hvor lang tid der er tilbage, før opgaven er færdig. Disse tidsestimater vist på 'Opslagstavlen' på figur 5.1.



**Figur 5.1:** Opslagstavlen

Alle opgaver har en livscyklus der starter med, at opgaven oprettes og placeres i kolonnen 'To do - skal laves'.

Når en opgave påbegyndes flyttes den til 'In progress - er i gang' for at fortælle gruppen, at opgaven er igangsat, og hvilket gruppemedlem der er i gang med opgaven.

Når opgaven er udført, flyttes den fra 'In progress - er i gang' til 'to verify - rettes' hvor opgaven forbliver, indtil et andet gruppemedlem har gennemlæst eller på anden måde kigget på opgaven.

Til sidst placeres opgaven i 'Done - færdig' for at markere opgaven er færdig. Dette gøres ofte efter en eller flere gruppemedlemmer har været gennem 'to verify - rettes'. Opgaven beholdes, så det senere er muligt at finde ud af, hvilke opgaver der blev løst af hvem og hvornår.

## 5.4 Hvad vi har lært

En ting der tages med fra denne læringsproces er, at udnytte taskboardet til fulde. Med dette menes der, at der har manglet en form for projektleder, som skulle stå for at opdatere taskboardet. Dette har betydet, at der har været tilføjet opgaver hist og her, og nogle er mere meningsfyldte end andre. De agile dele af udviklingen kunne være undgået, såfremt vi brugte taskboardet slavisk. I nogle situationer er noget blevet færdigt, før det overhovedet nåede til taskboardet. Dette skyldes, at det har ligget tæt op ad en opgave som var ved at blive udført, og derfor var nem

at lave samtidig. Ved at have en projektleder der stod for taskboardet, opdateringer og opgavedeling, er det muligt at optimere iterationer og tidsforbrug.

Dette projekt er en læringsproces i forhold til at udvikle et system sammen med en virksomhed. Der blev givet en grænse på mindst én use-case, men ingen øvre grænse. Vi har erfaret, at man bør sætte sig et realistisk mål og arbejde mod det, fremfor at forsøge at nå så meget som muligt. Dette kan også være en af grundene til at UP-processen til tider har båret præg af agil udviklingsproces, da vi gerne ville nå at udvikle så meget som muligt.

Det er væsentligt, at når man snakker med en virksomhed at få afstemt forventninger, og at man er sikker på, hvad der udvikles efter. Det blev oplevet, at Brovst Bolighus skiftede idé, om hvad der kunne udvikles. Det kan tænkes, at der i denne proces blev vist et omfang af, hvad der kunne udvikles for virksomheden, og kunden derved fik nye ideér ved det seneste review. Det er derfor også erfaret, at man kan lave en "kontrakt", hvor udviklerne og virksomheden laver en forventningsafstemning.



## 6 | Konklusion

I dette kapitel konkluderes der, hvorvidt problemformuleringen er opfyldt.

Problemformuleringen til systemet er, som nævnt i indledningen:

*Hvordan kan der udvikles et IT-system, som gør det nemmere for Brovst Bolighus at håndtere bestilling af specialdesignede møbler, finde ud af hvor der skal markedsføres og lettere finde varer på lageret?*

Systemet opfylder kun funktionaliteten i forhold til at oprette en salgsordre af specialdesignede møbler. Det betyder, at problemformuleringen kun delvist opfyldes, da håndtering af specialdesignede møbler ikke er færdig-implementeret. Der er heller ikke designet noget til markedsførings og lagerdelsystemerne.

Planen med systemet er, at det skal fungere som et samlet ERP-system til Brovst Bolighus. For at dette kan lade sig gøre, skal der fremadrettet implementeres resten af funktionaliteten i forhold til ordrer og de to andre delsystemer.

Systemet er delvist fremtidssikret i forhold til at Brovst Bolighus udvider deres varesortiment med nye produktkategorier.



# Litteratur

- [1] Hans Jørgen Skriver, Erik Staunstrup, and Peter Storm-Henningsen. *Organisation*. Trojka / Gads Forlag A/S, 2012. ISBN 978-87-92098-65-8.
- [2] Danbo Møbler. De bedste priser på møbler, 2017. URL <http://danbomoebler.dk/upload/Brovst/moebler-thisted-aalborg.html>.
- [3] Postnord. Antal modtagere pr. 19-02-2017, 2017. URL [http://www2.postdanmark.dk/iis-adresselose\\_2012/modtagere.asp](http://www2.postdanmark.dk/iis-adresselose_2012/modtagere.asp).
- [4] Kapitel 2 - Logistik - mål og strategier, 2013. URL [https://ucn.instructure.com/courses/7801/files/275178?module\\_item\\_id=104314](https://ucn.instructure.com/courses/7801/files/275178?module_item_id=104314).
- [5] *Programmering slides fra lektion 1, andet semester*, 2017. URL <https://ucn.instructure.com/courses/7799/modules/items/116909>.
- [6] *Systemudvikling slides fra lektion 5, andet semester*, 2017. URL <https://ucn.instructure.com/courses/7800/modules/items/128971>.
- [7] *Systemudvikling slides fra lektion 6, andet semester*, 2017. URL <https://ucn.instructure.com/courses/7800/modules/items/131747>.
- [8] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, Third Edition*. Addison Wesley Professional, 2004. ISBN 0-13-148906-2.
- [9] KOMP s. 62-73 Costbenefit - udleveret af Lise Klitsgaard, 2016. URL [https://ucn.instructure.com/courses/7791/files/260418?module\\_item\\_id=98453](https://ucn.instructure.com/courses/7791/files/260418?module_item_id=98453).
- [10] *Systemudvikling slides fra lektion 1, første semester*, 2016. URL <https://ucn.instructure.com/courses/7790/modules/items/73454>.
- [11] *Systemudvikling slides fra lektion 2, første semester*, 2016. URL <https://ucn.instructure.com/courses/7790/modules/items/73572>.

- [12] Jakob Nielsen. *Usability 101: Introduction to Usability*, 2012. URL <https://www.nngroup.com/articles/usability-101-introduction-to-usability/>.
- [13] *Gestaltlovene*, 2016. URL <http://www.nielsgamborg.dk/?p=gestaltlovene>.
- [14] Niels Gamborg. *Loven om figur og baggrund*, 2016. URL <http://www.nielsgamborg.dk/?p=gestaltlovene&u=figur>.
- [15] Niels Gamborg. *Loven om nærhed*, 2016. URL <http://www.nielsgamborg.dk/?p=gestaltlovene&u=naerhed>.
- [16] Niels Gamborg. *Loven om lighed*, 2016. URL <http://www.nielsgamborg.dk/?p=gestaltlovene&u=lighed>.
- [17] Niels Gamborg. *Loven om lukkethed*, 2016. URL <http://www.nielsgamborg.dk/?p=gestaltlovene&u=lukkethed>.
- [18] Niels Gamborg. *Loven om forbundethed*, 2016. URL <http://www.nielsgamborg.dk/?p=gestaltlovene&u=forbundethed>.
- [19] Jakob Nielsen. *10 Usability Heuristics for User Interface Design*, 1995. URL <https://www.nngroup.com/articles/ten-usability-heuristics/>.



# A | Metode

I dette kapitel vil der være en kort beskrivelse og forklaring af de metoder og teorier der benyttes i rapporten.

## A.1 Kotter

Kotters 8-trins forandringsmodel	
Hovedfaser (Lewin)	Trin (Kotter)
Fase 1: Optøning	1. Etablere en oplevelse af nødvendighed 2. Oprettelse af den styrende koalition 3. Udvikling af en forandringsvision 4. Formidling af forandringsvisionen
Fase 2: Forandring	5. Gøre forandringen mulig 6. Generering af kortsigtede gevinster 7. Konsolidering af resultater og produktion af mere forandring
Fase 3: Fastfrysning	8. Forankring af nye arbejdsmåder i kulturen

Fig. 11.18 Kotters otte-trins proces sammenholdt med Lewins 3 faser.

Figur A.1: [1, s.407-410]

Punkt 1:

Hvis selvtilfredsheden er stor i virksomheden, kan forandring være svært at komme i gang med. Det kan være at der mangler åbenhed i virksomheden, og at medarbejderne skal forstå, at hvis denne forandring ikke gennemføres kan det betyde, at virksomheden afvikles.

Punkt 2:

Det er vigtigt, at der i virksomheden oprettes et stærkt team der:

- Har de nødvendige kompetencer.
- Fornødene troværdighed.
- Ledererfaring og magt i virksomheden.

Det stærke team er den gruppe der skal lede organisationen ud af en krise og skabe trykthed.

Punkt 3:

Gruppen skal ikke udvikle planer, men visioner og strategier som kan styre forandringsprocessen. Dette er vigtigt da forandringsprocessen angiver retning og formål med forandring.

Punkt 4:

En stærk vision for forandring er uden nytte, hvis ingen i organisationen kender til den.

Det er gruppes opgave at skabe og bruge kommunikationskanaler, der er til rådighed. Kommunikation og information skal komme løbende, og opdateringer er vigtige.

Punkt 5:

Medarbejdere i organisationen skal være rustede til de nye forandringer. De skal derfor efteruddannes eller på kursus for at få de fornødne færdigheder. Stærke modsætninger til forandringen skal fjernes, selv hvis dette betyder fyringer.

Punkt 6: Kortsigtede gevinster har til formål at skabe tillid til den styrende enhed, og til at lede forandringsprocessen. Det er vigtigt at huske, at store forandringer tager tid. Medarbejder i organisationen skal se beviser for at retningen er den rigtige og inkluderes i gevinsterne.

Punkt 7:

Når de kortsigtede gevinster er skabt, og hvis motivationen og indsatsen findes hos medarbejderne, er det vigtigt at vedligeholde disse resultater ved at indbringe nye eller flere ressourcer for at bevare den positive stemning. Hermed kan der sættes gang i flere forandringer, hvis den ledene gruppe kan håndtere disse.

Punkt 8:

Forandring kan først rigtig forankres når punkt 3 er opfyldt, og medarbejdernes arbejdsvaner er ændret, til den nye vision. Hvis det ikke er muligt at ændre de ansattes vaner og rutiner, er det ikke muligt at implementere forandringen. Derfor bør ledelsen starte med at ændre normer og værdier, og være opmærksom på at

virksomhedskulturen først ændres inden forandringen er på plads.

## A.2 McGregor's X- og Y-teori

Ved en leders menneskesyn forstås, hvordan en leder opfatter medarbejderne. Ifølge McGregor findes der overordnet to menneskesyn, som han kalder for X- og Y-syn. På figur A.2 ses, hvilke karakteristika der definerer disse to menneskesyn [1, s. 205].

McGregors X- og Y-syn		
Lederens antagelser	X-syn	Y-syn
Hvilket forhold har de til det at arbejde?	Føler ubehag ved at arbejde, og forsøger derfor at undgå det	Arbejde er lige så naturligt for mennesket som fx leg og hvile
Hvilke ønsker har de til arbejdets indhold?	Foretrækker at blive dirigeret til at udføre simple rutine-opgaver	Frihedsgrader til at udføre udviklende arbejdsopgaver sådan, at medarbejderen kan styre sig selv i retning mod mål, som han selv går ind for
Hvilke behov ønsker de at få dækket via arbejdet?	Tryghed og sikkerhed samt de fysiske behov	Ego og selvrealisering ved at udvikle sig via jobbet
Er de villige til at påtage sig et ansvar?	Nej – vil hellere have tryghed ved at andre tager ansvaret	Ja – hvis det er i forbindelse med et meningsfyldt arbejde

Figur A.2: McGregors X- og Y-syn [1, s. 205].

## A.3 Adizes lederroller

Adizes teori bygger på hvilke personlige egenskaber der er nødvendige for bestemte opgaver, så den enkle opgave kan løses af ledelsen. Ifølge Adizes har en effektiv ledelse fire overordnede roller.

De fire roller Adizes bruger er:

- Producentrollen
- Administratorrollen
- Entrepreneurrollen
- Integratortrollen

En oversigt over disse rollers egenskaber kan ses på figur A.3.

		Fokuserer på	
		Produkt	Proces
Fokuserer primært på	Langt sigt	<b>Entrepreneurrollen</b> <ul style="list-style-type: none"> <li>• Kreativ og innovativ</li> <li>• Finder nye produkter og nye metoder</li> <li>• Tænker strategisk</li> <li>• Stiller spørgsmål til det bestående</li> <li>• Risikovillig</li> <li>• Udvikling</li> </ul>	<b>Integratorrollen</b> <ul style="list-style-type: none"> <li>• Integrerer i et fællesskab</li> <li>• Indgår kompromiser</li> <li>• Skaber motivation og korpsånd</li> <li>• Leder gennem teamwork</li> <li>• Skaber udvikling hos medarbejderne</li> <li>• Skaber sammenhold</li> </ul>
	Kort sigt	<b>Producentrollen</b> <ul style="list-style-type: none"> <li>• Resultat- og handlingsorienteret</li> <li>• Stort præstationsbehov</li> <li>• Tager beslutninger</li> <li>• Flittig og travl</li> <li>• Medarbejderne bliver hjælpere</li> <li>• Faglig viden</li> </ul>	<b>Administratorrollen</b> <ul style="list-style-type: none"> <li>• Opstiller mål og regler</li> <li>• Kontrollerer og evaluerer</li> <li>• Skaber systematik</li> <li>• Analyserer sig frem til den rigtige løsning</li> <li>• Bureaukrati</li> <li>• Ordenssans</li> </ul>

Figur A.3: Adizes' 4 lederroller [1, s. 208].

For at kunne tildele "roller" til den enkelte leder bruges der et *kodesystem* med tre niveauer.

- Stort bogstav betyder stor varetagelse af rollen
- Lille bogstav betyder nogen varetagelse af rollen
- Et O betyder, at lederen overhovedet ikke dækker rollen

Ud fra dette kan der tildeles fire bogstaver til en leder, eks. PAEI, PaEi, eller pOeI. Det er som regel umuligt at have store bogstaver ved alle rollerne, da nogle er i strid med hinanden. Dog skulle en leder kunne opfylde hver rolle i et rimeligt omfang, ellers er der ifølge Adizes tale om *mismanagement* [1, s. 207-209].

## A.4 SWOT-analyse

En SWOT-analyse bruges til at samle informationer fra andre foregående analyser og observationer. Måden en SWOT-analyse er bygget op på, er ved at inddele virksomheden i fire dele, hvoraf to er interne forhold og to er eksterne forhold. Det er dog værd at bemærke, at SWOT-analysen kun giver et indblik i, hvordan den nuværende situation er for virksomheden, men ikke hvordan situationen vil være i fremtiden.

De interne forhold består af stærke og svage sider, og giver et overblik over, hvilke styrker og svagheder virksomheden har. De interne forhold karakteriseres desuden

ved, at det er noget, som virksomheden selv har indflydelse på.

De eksterne forhold består af muligheder og trusler. Disse bruges til at give et øjebliksbillede af de elementer virksomheden har mulighed for at udnytte og hvilke konkurrencemæssige udfordringer de bør tage højde for [1, s. 362-363].

Et eksempel på hvordan en SWOT-analyse kan sættes op er vist på figur A.4.

Interne forhold	
Stærke sider (Strengths)	Svage sider (Weaknesses)
Eksterne forhold	
Muligheder (Opportunities)	Trusler (Threats)

Figur A.4: SWOT eksempel

## A.5 Ansoffs vækstmatrix

Ansoffs vækstmatrix er et redskab udviklet af Igor Ansoff som bruges til at systematisere en virksomheds mulige vækststrategier. Modellen som vises på figur A.5, inddeles i fire mulige strategier for at opnå den ønskede vækst. Yderligere skelner modellen mellem nuværende og nye markeder og produkter [1, s. 370-371]:

- Markedspenetrering
  - Ved denne strategi søger virksomheden at sikre et øget salg af eksisterende produkter på det nuværende marked ved at fortrænge konkurrenterne.
- Markedsudvikling
  - Denne strategi betegner, at virksomheden forsøger at afsætte sine nuværende produkter til nye markeder.
- Produktudvikling
  - Ved produktudvikling sker der en udvikling af nye varer til eksisterende markeder. Når denne strategi bruges, tilstræber virksomheden at udnytte "first mover-effekten", som handler om at være den første med et nyt produkt.
- Diversifikation
  - Ved diversifikation udvikles nye produkter til helt nye kundegrupper og markeder. Det vil sige, at virksomheden ikke kun bevæger sig ind på et nyt marked, men også med et helt nyt produkt.

Ansoffs vækstmatrix			
		Produkter	
		Nuværende	Nye
Markeder	Nuværende markeder	Markedspenetrering	Produktudvikling
	Nye markeder	Markedsudvikling	Diversifikation

Figur A.5: Ansoffs vækstmatrice [1, s. 370].

## A.6 Information Economics

Information Economics benyttes til at uddybe den traditionelle cost-benefit analyse. Dette gøres ved at eksplicit evaluere IT-projekter med henblik på at vægte forskellige faktorer imod hinanden.

Disse faktorer ses på figur A.6.

	Forretningsområde (Business Domain)	IT – område (IT – domain)
Positive faktorer, der understøtter valgmuligheder mellem de udvalgte projekter	Strategic match Competitive advantage Management information Competitive response	Strategic IS architecture
Negative faktorer, der forringer valgmuligheder mellem de udvalgte projekter	Organisational risk	Definition uncertainty Technical uncertainty IS infrastructure risk

Figur A.6: Diverse faktorer som tages i betragtning [9, s. 3]

Som det fremgår af figuren er der både positive og negative faktorer der spiller ind i forhold til de udvalgte projekter. De positive repræsenterer værdi for virksomheden og de negative faktorer er risici eller usikkerheder [9, s. 3].

Vægtene, vurderingerne, faktorerne og de mulige projekter opstilles i en tabel som ses i figur A.7. Delsystemet med den højeste total, er det delsystem som bør være det vigtigste for virksomheden.

	Business domain						Technology domain				To tal
	ROI	SM	CA	MI	CR	OR	SA	DU	TU	IR	
Vægt	+10	+2	+2	+2	+1	-1	+3	-2	-2	-2	
Debitorstyring	40	8	4	6	2	0	3	0	0	0	63
E-handelssystem	30	10	8	6	5	-4	6	-6	-6	-6	43

**Figur A.7:** Diverse faktorer som vurderes ud af vigtighed [9, s. 10]

De faktorer der har at gøre med forretningsområdet er ROI, SM, CA, MI og CR. Beskrivelse af disse ses i tabel A.1.

ROI	Return of investment relaterer til cost-benefit analysen, og beskriver afkastet på investeringen.
SM	Strategic match beskriver i hvilken grad et projekt passer til virksomhedens strategiske målsætning.
CA	Competitive advantage bruges til at kvantificere i hvilken grad et projekt forbedrer virksomhedens position på markedet.
MI	Management information beskriver hvor meget projektet bidrager til ledelsens behov for information angående virksomhedens kerneområder, som eksempelvis stigende omsætning eller lavere salgsomkostninger.
CR	Competitive response bruges til at beskrive den forretningsrisiko der er ved at projektet ikke gennemføres. Der kan f.eks. være en risiko ved at virksomheden mister markedsandele der ikke kan generhverves.
OR	Organisational risk bruges til at bedømme hvor klar organisationen er til at gennemføre projektet.

**Tabel A.1:** Beskrivelse af faktorer der falder ind under forretningsområdet.

Faktorerne tilhørende teknologi er SA, DU, TU og IR. Disse beskrives på tabel A.2.

SA	Strategic IS architecture beskriver hvorvidt det er muligt at gennemføre projektet med eksisterende software og hardware.
DU	Definition uncertainty bruges til at værdisætte projektet i forhold til kendskabet af noget opstillede forudsætninger og krav.
TU	Technical uncertainty beskriver hvilke tekniske færdigheder den nye hard- og software kræver.
IR	IS infrastructure risk beskriver hvor meget organisationens infrastruktur skal ændres.

**Tabel A.2:** Beskrivelse af faktorer der falder ind under teknologiområdet.

## A.7 Forandringsstrategier

En forandringsstrategi kendetegnes ved at der overvejes, hvilke indsatsområder og metoder man vil anvende i forhold til at ændre et system, samt hvilken type af forandring der er tale om. I det følgende opstilles forudsætninger for fire forskellige forandringsstrategier [1, s. 397-399].

### Ekspertstrategi

Ved denne strategi benyttes en eller flere eksperter til at stille en diagnose eller analysere et problem. Diagnosen eller analysen ender ud i en problemformulering eller/og et design skræddersyet til systemet.

### Socioteknisk strategi

Denne metode bruges, når der skal opnåes kompromis mellem evt. interne modstridende interesser og ved udformning af systemer. Der ansættes konsulenter til at interviewes, finde løsningsforslag og finde viden fra brugere.

### Repræsentationsstrategi

I modsætning til ekspertstrategien, hvor få udvalgte er med til at forme systemet, benytter Socioteknikken 'almindelige' medarbejdere. Ud fra disse medarbejdere stilles repræsentanter, som forsøger at tilføje sociale mål i systemet.

### Deltagelsesstrategi

Ved deltagelsesstrategi bruges de egentlig slutbrugere, som ved de andre metoder ikke har megen indflydelse. Dette gør at der her tages større hensyn til slutbrugernes behov til systemet.

I figur A.8 ses en visuel fremvisning af diverse forandringsstrategier.

Forandringsstrategier i forhold til menneskesyn og organisationsopfattelse			
Menneskeopfattelse			
		X-teori	Y-teori
		Ekspertstrategi	Repræsentationsstrategi
Organisationsopfattelse	Harmoni		
	Konflikt	Socioteknisk strategi	Deltagelsesstrategi

Figur A.8: Forandringsstrategier i forhold til organisationsopfattelse og menneskesyn [1, s. 401]



## A.8 Medarbejder - opgave - mål tabel

Medarbejder - opgave - mål tabel benyttes til, at opstille hvordan en "opgave" udføres ud fra en medarbejders perspektiv. Dette gøres ved, at følge hvordan medarbejderne udfører opgaven igennem "steps" i alle tænkelige use-cases [8, s.77-78].

Disse opgaver og mål identificeres, ved at tage udgangspunkt i IT-forundersøgelsen eller selve interviewet med kunden. Ud fra dette identificeres kundens nuværende opgaver og de målbare resultater[10].

Resultatet af disse use-cases opstilles i en medarbejder - opgave - mål tabel, som vist på eksemplet i tabel A.3.

Medarbejder	Opgave	Mål	Step i opgave
Ekspedient	Modtage ny ordre	Registrer ordre	Finde varer kunde ønsker Tilføje varerne til ordren Tilføje kundeoplysninger Færdiggøre ordren Bekræftelse af ordren
...	...	...	...

Tabel A.3: Eksempel på en medarbejder - opgave - mål tabel

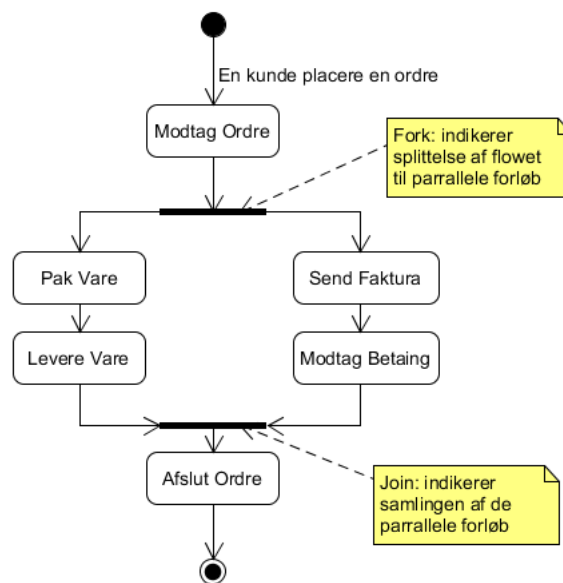
## A.9 Workflow

Et workflow diagram er et UML modelleringsværktøj, som viser sekvensen af handlinger forbundet med en use-case. Et workflow kan anvendes til ethvert formål, dog ses det ekstra brugbart når det bruges i sammenhæng med enten *forretningsmodellering*, *en process* eller en *use-case* [8, s. 607].

- Forretningsmodellering
  - Bruges til at give et overblik over en forretningsstruktur, samt dynamikkerne i en organisation med henblik på at indfører et nyt IT-system.
- Objektmodellering
  - Bruges til at give en visuel repræsentation over virksomhedsstruktur. Dette gøres ved at benytte *UML klasser*, *sekvenser*, og *aktivitetsdiagrammer*.
- Use-case
  - Bruges til at give en visuel repræsentation over hvordan en handlinger udføres i en use-case. Dette kan foregå parallel med hinanden, hvorved

flowet splittes med en "fork". Slutteligt vil de parallelle handlingsforløb samles med et "join" og opgaven indikeres afsluttet ved, at flowet når den hule cirkel. Typisk vil medarbejder - opgave - mål tabellen danne baggrund for et workflowdiagram, som også kan betragtes som en visualisering af samme.

Et eksempel på et workflow diagram som tager udgangspunkt i en use-case, ses på figur A.9.



Figur A.9: Eksempel på et workflow.

## A.10 Use-case

En use-case beskriver en række handlinger, som bliver udført af en aktør for at udføre et bestemt mål. Use-cases beskrives med henblik på at identificere, klarlægge og organisere krav til systemet samt finde ud af, hvordan aktøren interagerer med systemet [11].

### A.10.1 Use-case beskrivelser

I rapporten bruges der to former for beskrivelser af use-cases. Den korte form, brief, og den mere beskrivende, som kaldes fully dressed.

### Brief

En brief beskrevet use-case beskrives overordnet men komplet. Dette vil sige, at der beskrives, hvem der initierer use-casen, de forventede systemhandlinger og responsen herpå, som giver systemet værdi for aktøren [11].

### Fully-dressed

Fully dressed use-cases beskriver, hvordan en opgave udføres med henblik på at nå til mål med use-casen. Der bliver benyttet to forskellige stadier i en fully dressed: *Happy days* og *alternative flows*. Happy days beskriver, hvordan opgaven udføres, når alle forudsætninger for opgaven er som det forventes. Alternative flows beskriver situationer, som forhindrer direkte udførelse af opgaven.

En fully dressed beskrivelse kommer både med en præ- og en postbetingelse. Præbetingelsen beskriver hvilke betingelser, der skal være opfyldt før handlingen er mulig, og postbetingelsen beskriver, hvad der skal være opfyldt [8, s. 126].

### A.10.2 Use-case diagram

Use-case diagrammer er en illustration af hvilke aktører der interagerer med udførte use-cases. Dette gøres, for at skabe overblik over, hvem der skal hvad og hvilke funktioner de skal have adgang til [8, s. 158].

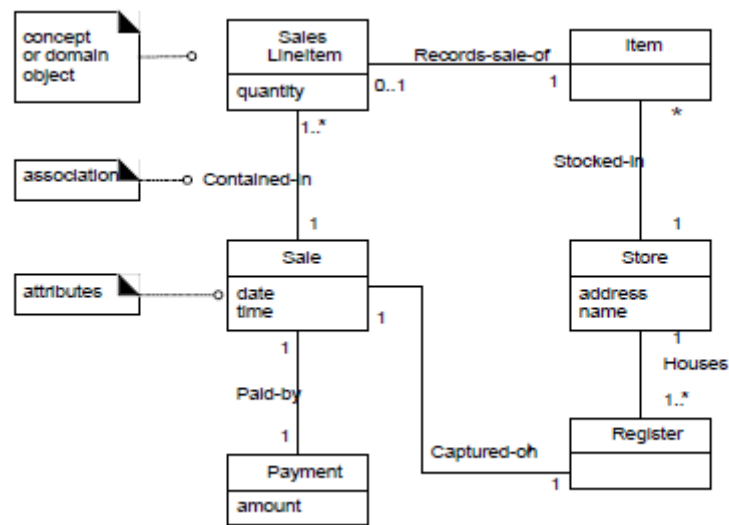
## A.11 Domænemodel

Domænemodeller er en repræsentation af konceptuelle klasser, der benyttes som et analyseredskab i systemudvikling [8, s. 128].

En domænemodel kan vises ved hjælp af et klassediagram som indeholder:

- attributter tilhørende klasserne
- konceptuelle klasser
- association mellem klasserne

Figur A.10 viser et eksempel på en domænemodel hvor det kan ses, hvordan klasser, attributter og associering mellem klasser er planlagt i systemet.



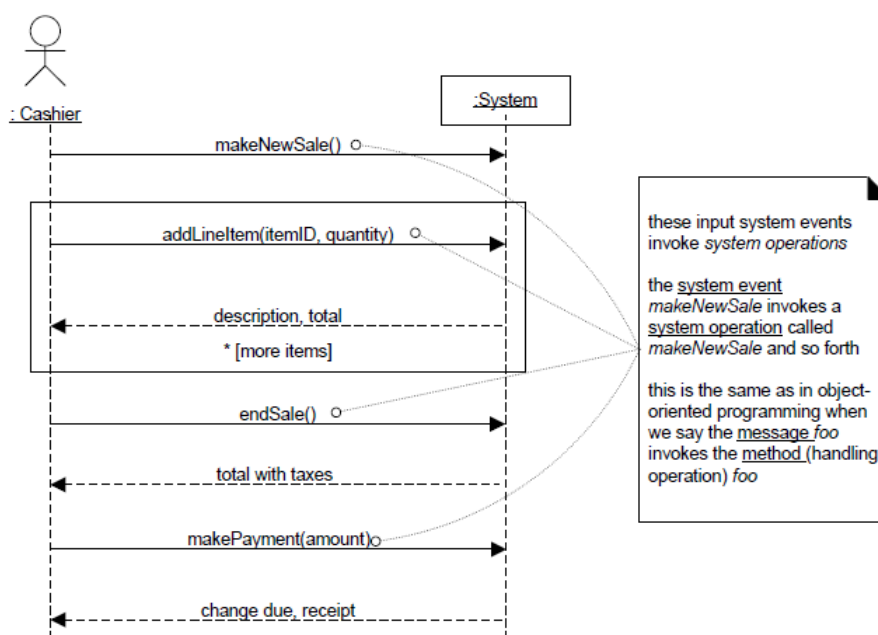
Figur A.10: Domænemodel eksempel [8, s. 129]

## A.12 System kontrakter

Kontrakter beskriver detaljeret de ændringer i stadier de forskellige objekter gennemgår igennem domænemodellen, efter et forløb er udarbejdet.[8, s.177]

### A.12.1 System-sekvensdiagrammer

*System-sekvensdiagrammer (SSD)* er et sekvensdiagram, som viser de handlinger som en aktør udfører, og det tilhørende system feedback. Systemet ses for at være en "Black box" som behandler de udførelser som aktøren laver. Diagrammet skifter derfor mellem aktør og system[8, s. 178].

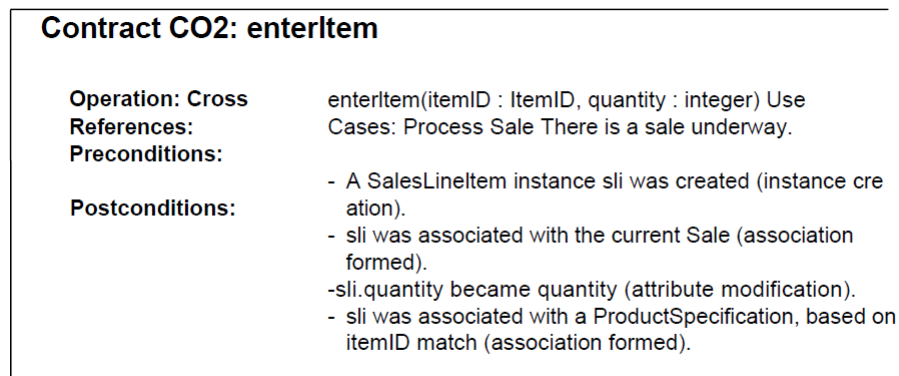


Figur A.11: System-sekvensdiagram eksempel [8, s. 178].

### A.12.2 Operationskontrakt

*OperationsKontrakt* er en kontrakt, med en *præ* og *post-betingelse*, dette gøres for at vise stadiet på objekterne før use-casen bliver udført, og hvordan de forskellige handlinger igennem use-casen påvirker stadierne på objekterne [8, s. 179 - 190].

- Præ betingelse
  - Fortæller de krav som bør være kendt før use-casen starter.
- Post betingelse
  - Post betingelser er ikke operationer som foretages i løbet af use-casen, men stadier omkring domænemodellens objekter når use-casen er udført.



Figur A.12: Operationskontrakt eksempel [8, s. 179]

### A.12.3 Kommunikationsdiagram

Kommunikations diagrammer illustere interaktioner imellem objekter i enten en graf eller et netværksformat. I et kommunikationsdiagram kan der vertikalt udtrykkes nye objekter, da objekter ikke har nogen fast plads. Dette gør at, der kan skrives

## B | Cost-benefit beregninger

HP MicroServer Gen8 Entry	1699,45	<a href="http://www.bj-trading.dk/bjshop/default.asp?pv=819185-421&amp;pn=Hewlett-Packard&amp;gruppe=Zedbpri&amp;overgr=Andet&amp;vare=992220&amp;f=edbp">http://www.bj-trading.dk/bjshop/default.asp?pv=819185-421&amp;pn=Hewlett-Packard&amp;gruppe=Zedbpri&amp;overgr=Andet&amp;vare=992220&amp;f=edbp</a>
3TB WD Red WD30EFRX	906,45	<a href="http://www.bj-trading.dk/bjshop/default.asp?pv=WD30EFRX&amp;pn=Western-Digital&amp;gruppe=harddisk-ide(SATA%203,5)&amp;overgr=Hardware&amp;vare=803889&amp;f=edbp">http://www.bj-trading.dk/bjshop/default.asp?pv=WD30EFRX&amp;pn=Western-Digital&amp;gruppe=harddisk-ide(SATA%203,5)&amp;overgr=Hardware&amp;vare=803889&amp;f=edbp</a>
	2605,90	DKK

**Tabel B.1:** Pris for server og harddisk

Apparat	150	watt
Tidsforbrug	24	timer om dagen
El-pris inkl. moms og afgifter	2,29	DKK pr. kWh
Specificerede priser		
Pris pr. dag	8,244	DKK
Pris pr. måned	247,32	DKK
Pris pr. år	3009,06	DKK
Samlet kWh på et år	1314	kWh

**Tabel B.2:** Pris for strøm

Udviklingsomkostninger	Ordrehåndteringsdelsystem	Markedsføringsdelsystem	Lager-delsystem
Løn for en datamatiker pr. måned (DKK)	32504	<a href="https://www.prosa.dk/raadgivning/loenstatistik/startloenninger/">https://www.prosa.dk/raadgivning/loenstatistik/startloenninger/</a>	
Antal måneder	15	6	0,5
Lønningsomkostninger (DKK)	487560	195024	16252

**Tabel B.3:** Udviklingsomkostninger

Løn for en datamatiker pr. time (DKK)	450		
Antal timer pr. måned	5	3	1
Antal timer på et år	60	36	12
Lønningsomkostninger (DKK)	27000	16200	5400

**Tabel B.4:** Support og vedligeholdelse

Løn for en ansat pr. time (DKK)	150
Antal timer pr. måned	20,67
Antal timer på et år	248
Lønningsomkostninger (DKK)	37200

**Tabel B.5:** Sparet løn ved bestilling

Løn for en ansat pr. time (DKK)	150
Antal timer pr. måned	18,6
Antal timer på et år	223,2
Lønningsomkostninger (DKK)	33480

**Tabel B.6:** Sparet løn ved at kigge på lager



Salg til flere kunder	Ordrehåndterings- delsystem	Markedsførings- delsystem	Lager-delsystem
Bruttofortjeneste 2016 (DKK)	2829000		
Forventet stigning (%)	10	30	0
Forventet merindtjening (DKK)	282900	848700	0

Tabel B.7: Salg til flere kunder



## C | Parker Benson

Technical uncertainty	Ordrehåndterings delsystem	Markedsførings delsystem	Lagerstyrings delsystem
	Score	Score	Score
A: Er de nødvendige færdigheder tilstede i det tekniske område	3	3	3
B: Afhængighed af specifikt hardware er ikke til stede i øjeblikket	0	0	0
C: Afhængighed af software (andet end application software)	1	1	1
D: Afhængighed af anvendelsen	3	3	1
Total Rating: (A+B+C+D) / 4	1.75	1.75	1.25

**Figur C.1:** Beregning for technical uncertainty til cost-benefit analyse.



# D | Brugervenlighed

I dette kapitel gennemgås teorien for hvordan et system designes, sådan at det er brugervenligt. Afsnittet tager udgangspunkt i Jakob Nielsens 10 regler om design af en brugergrænseflade.

## Teori

Brugervenlighed er en essentiel del af enhver datalogisk løsning, da dette gør, at en bruger uden meget datalogisk viden skulle kunne benytte sig af en datalogisk løsning, samt at dette har vist sig at være tidsbesparende ved udførelse af diverse opgaver som brugeren udfører. *Nielsen Norman Group* har defineret hvordan brugervenlighed implementeres sådan det bedst muligt kan benyttes af flere forskellige brugergrupper [12]. Disse fem komponenter er defineret som følgende:

- Learnability
  - Definer hvorvidt en bruger kan udfører en opgave første gang designet benyttes.
- Efficiency
  - Definer hvor hurtigt en bruger kan udfører opgaver, efter brugeren har stillet kendskab til designet.
- Memorability
  - Definer hvor hurtigt en bruger kan benytte designet effektivt efter, at have været fraværne fra designet.
- Errors
  - Definer hvorvidt en bruger kan overkomme fejl i systemet, og at fejlhåndteringen skal foregå på almindeligt sprog.
- Satisfaction
  - Definer tilfredsheden en bruger har ved benyttelse af systemet.

Disse fem komponenter giver et overordnet overblik af et design i en datalogisk løsning, dog er det ikke defineret hvordan den gennemsnitlige bruger ser den logiske sammenhæng i et design af en brugergrænseflade. Dette kastes der et blik over i næste underafsnit.

## Gestaltlovene

I dette underafsnit beskrives gestaltlovene, dette gøres for at give et overblik over hvordan den gennemsnitlige bruger opfatter en logisk sammensætning af forskellige visualiteter.

Gestaltlovene består af fem love, disse love benyttes til at give et blik over hvordan en brugergrænseflade skal designes, ud fra hvordan mennesket sanser sammenhængen i et design. Gestaltlovene er udarbejdet igennem videnskabelige forsøg, hvorfra der blev testet hvordan de fleste mennesker, logisk forstår sammenhæng imellem visuelle objekter [13]. Disse fem regler er følgende:

- Loven om figur og baggrund
  - Beskriver at der skal defineres præcist hvilke visuelle dele som er for- og baggrund, således det ikke skaber forvirring i forståelsen for hvilke dele som er de centrale [14].
- Loven om nærhed
  - Definer sammenhængen imellem forskellige visuelle objekter, dette gøres hos brugeren ubevidst, og derfor bør det klargøres hvilke visuelle objekter som har relevans til hinanden [15].
- Loven om lighed
  - Definer at ligheden imellem designet og opstilling af form, placering, størrelse og farve bør være ens og konsekvent hele designet igennem [16].
- Loven om lukkethed
  - Beskriver hvordan de forskellige handlinger skal afsluttes, Dette gøres ved at give klar feedback i forskellige former af visuelle bokse, rammer og tekster sådan der skabes overblik over hvilke handlinger som er udføres [17].
- Loven om forbundethed
  - Definer hvordan sammenhængen imellem de visuelle objekter kan vises, dette kan foregå ved klar sammenkobling med f.eks. linjer imellem objekterne, at farvede flader forbinder hinanden, eller med overlapning imellem objekterne [18].

Gestaltlovene giver et indblik i hvordan den gennemsnitlig bruger, forstår den

logiske sammenhæng i et design. Ud fra dette kastes der et blik over hvordan et bestemt design af et *user interface* skal designes, dette gøres ved hjælp af Jakob Nielsens 10 regler.

## De 10 regler

I denne sektion tages der udgangspunkt i Jakob Nielsens 10 *heuristikker* ved design af en brugergrænseflade. Disse regler kaldes heuristikker fordi de er overordnede regler og ikke specifikke retningslinjer som bør følges til punkt og prikke [19].

1. Giv feedback ved udførelse af handlinger.
2. Systemet skal ikke benytte system-orienteret termer.
3. Brugeren skal have kontrol over systemet.
4. Stræb efter konsistens.
5. Simpel fejlhåndtering.
6. Systemet skal holde information fra tidligere handlinger.
7. Systemet skal være fleksibelt og hurtigt.
8. Simpel og konkret feedback.
9. Fejlmeddelelser skal ikke indholde tekniske termer.
10. Nem adgang til hjælp.

Disse 10 regler giver et godt overblik over hvordan en brugergrænseflade skal designes, sådan enhver bruger kan forholde sig til de funktionelle elementer i designet.

## Opsummering

Med Gestaltlovene, afsnit D, og de 10 heuristikker, som er beskrevet i afsnit D, er der her opstillet nogle klare retningslinjer, som kan bruges til at designe et *user interface*, således det er forståeligt for den gennemsnitlige bruger med henblik på både det visuelle og det funktionelle.





# E | Usability test

Dato for rapport: 08-05-2017

Dato for test: 08-05-2017

Lokation: UCN Sofiendalsvej, Nordjylland

Forberedt af:

Casper Froberg Andersen

Tobias Andersen

Stefan Krabbe Johansen

Mikkel Lindstrøm Paulsen

Arne George Ralston

## Resume

Denne rapport beskriver et ekspert reviewt af mock-ups som er lavet i forbindelse med at designe et user Interface til programmet SmartOrder, som er et ordre program der udvikles til Brovst Bolighus ApS. Der er udarbejdet testscenarier til mock-ups, som de 2 testpersoner vi har fået tildelt skal igennem. Det skal dog nævnes at da det blot er mock-ups, og en del af designprocessen, så er programmet ikke interaktivt, og derfor foregår valg og overvejelser mundtligt. Det betyder også at opgaverne er lidt mere simple, end de ville være såfremt det var et fuld interaktivt program. De forskellige scenarier er beskrevet herunder i rapporten.

Testen blev udført som en tænke-højt-test, hvor deltagerne blev bedt om at sige hvad de tænkte om det der først faldt dem ind.

## Metode

### Hvem vi testede

Vi testede vores mock-ups til programmet SmartOrder med 2 deltagere, som begge studere til datamatiker på andet semester, og derfor har kendskab til gui design.

De havde følgende karakteristikker:

Alder		Computerbrug	
18-25	2	11 til 25 timer pr uge	2
Køn			
Mand	2		

Tabel E.1: Brugerbeskrivelse

## Førstehåndsindtryk af SmartOrder

Testperson 1:

- "Det ligner en skærm som en sælger bruger til at sælge varer."
- "Hyldevare er nok standard varer."
- "Specialdesign må være specialvarer"
  - Man skal vide hvordan modulerne kommer til at passer sammen.
- Testperson ser ikke at der er en søgefunktion i vinduet.

Testperson 2:

- "Det er et system til at lave ordrer"

## Indledende til ekspert review

### Hvad testpersonerne gjorde

Hver testperson brugte ca. 10 minutter til at komme gennem alle scenarier.

## Hvad data vi samlede

- 2/2 testpersoner kunne løse de første 4 scenarier uden udfordringer.
- 2/2 testpersoner havde udfordringer med scenarie 5. Udfordringer skal forstås som at opgaven ikke blev løst direkte, men blev dog løst mundtligt (da det også er mock-ups).
  - Testperson 1 sagde kun at man nok skulle kigge i ordre info.
  - Testperson 1 forsøgte dobbeltklik funktion, som der ikke er en implementation for, eller tiltænkt da mock-ups blev lavet.
  - Testpersonen kunne ikke umiddelbart se tilbud fra starten.

## Vigtige problemer som blev fundet og forslag til forbedringer

- Det er mock-ups, så det er svært at teste opgaverne helt som det er tiltænkt de skal løses.
- Ordretype er svært at finde, da der endnu ikke står noget i ordretype.
- Såfremt windows standard skal følges, skal de vigtigste og mest brugte knapper placeres på venstre side, og 'cancel' og lignende placeres i højre side.(eller længst nede)
- Ifølge testpersonen bør varenavn tilføjes til bunden ved de vare som er tilføjet.
- Boks i hjørne bør måske have en indikation på at det er ordre info, om den ordre man er på.
- Navigationsmenuen bør ikke være for overvældende. Man bør måske tænke over at have mindre kategorier.
- Begge testpersoner overså søgefunktionen. Denne bør være mere synlig.
- Vi bør se på om man skal have modulernes priser i systemet, når man vælger moduler.
- Testperson 1 synes 'Fjern vare' knappen er langt væk. Det er knappen som fjerner en vare fra en åben ordre.

## Positiv respons

Godt layout, overskueligt og placeringer af indhold giver god mening.

## Detaljerede testscenarier og forslag til forbedringer

### Om testen

- I "kommentarer" fra testpersonen, er quotes mellem anførselstegn.
- Alle testpersoner har inden testen fået følgende at vide:
  - Du kan ikke lave nogle fejl.
  - Vi gør dette for at teste et program for information og funktionalitet.
  - Vi tester ikke dig, men systemet.
  - Du må gerne tænke højt, når du navigere rundt i systemet.
  - Du må gerne være 100% ærlig

### Indledning - CreateOrderView vinduet vises

- Hvad tænker du at dette system skal bruges til?

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Det ligner et system til at håndtere nogle vare"</li> <li>• "Jeg kan bestille varer"</li> <li>• Hvad tror du boksen i venstre hjørne er til? (ordreinfo) - "Den er nok autogenereret"</li> <li>• "Man kan både finde kunde frem, og oprette kunde"</li> </ul>	<ul style="list-style-type: none"> <li>• Boks i hjørne bør måske have en indikation på at det er ordre info, om den ordre man er på.</li> </ul>

Tabel E.2: Kommentarer fra testperson 1

Kommentarer fra testperson 2	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Det ligner en skærm som en sælger bruger til at sælge varer"</li> <li>• "Hyldevarer må være almindelige standardvare"</li> <li>• "Specialdesign må være specielle varer"</li> <li>• Testpersonen er i tvivl om hvordan moduler passer ind.</li> </ul>	<ul style="list-style-type: none"> <li>• Begge testpersoner overså søgefunktionen. Denne bør være mere synlig.</li> </ul>

Tabel E.3: Kommentarer fra testperson 2

## Scenarie 1

Du vil tilføje en hyldevare til en ordre. Hvordan gør du dette?

Antal deltagere	2
Procent gennemført	100%

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• Testpersonen trykker på find varer.</li> <li>• Testpersonen mener at dropdown-funktionen er mærkelig fordi der ikke står noget der.</li> <li>• Testpersonen kan se at han er på hyldevarer, fordi den er markeret.</li> </ul>	<ul style="list-style-type: none"> <li>• Da det var mock-ups testen er udført på, er det forståeligt at det vil skabe lidt forvirring med tomme funktioner.</li> </ul>

Tabel E.4: Kommentarer fra testperson et ved scenarie et.

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• Testpersonen åbner hyldevare-tabben</li> <li>• Testpersonen markerer varen, skriver antal, og tilføjer varen til ordren.</li> </ul>	

Tabel E.5: Kommentarer fra testperson to ved scenarie et.

## Scenarie 2

Du vil tilføje en eksisterende kunde til en ordre. Hvordan gør du dette?

Antal deltagere	2
Procent gennemført	100%

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Jeg vil oprette en ny kunde hvis ikke jeg kan finde den pågældende kunde som eksisterende i databasen via søgefunktionen."</li> <li>• Testpersonen går ud fra kunde er tilføjet til ordren når felterne er fyldt med kundens information.</li> </ul>	<ul style="list-style-type: none"> <li>• Da det var mock-ups testen er udført på, er det forståeligt at det vil skabe lidt forvirring med tomme funktioner.</li> </ul>

Tabel E.6: Kommentarer fra testperson et ved scenarie to.

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Hvis kunden er i databasen, vil jeg søge på denne, ellers vil jeg oprette en ny kunde. Så i dette tilfælde ville jeg finde den eksisterende kunde."</li> </ul>	

Tabel E.7: Kommentarer fra testperson to ved scenarie to.

### Scenarie 3

Du vil se hvilke vare der er tilknyttet den ordre du arbejder med nu. Hvordan gør du det?

Antal deltagere	2
Procent gennemført	100%

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Man kan se i ordrelinjer hvilke varer der er tilføjet til ordren"</li> <li>• Testpersonen mener at varenavn bør tilføjes til bunden.</li> </ul>	<ul style="list-style-type: none"> <li>• Varenavn tilføjes til bunden ved de vare som er tilføjet</li> </ul>

Tabel E.8: Kommentarer fra testperson et ved scenarie tre.

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Umiddelbart kan det ses i ordre linjer"</li> </ul>	

Tabel E.9: Kommentarer fra testperson to ved scenarie tre.

## Scenarie 4

En kunde henvender sig for at bestille en specialdesignet vare. Hvordan opretter du denne ordre?

Antal deltagere	2
Procent gennemført	100%

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Man går ind på special design, og vælger en model"</li> <li>• "Man tilføjer derefter de ønskede moduler og kan tilføje den endelige sofa til ordren når man er færdig."</li> <li>• "Måske bør man overveje at have priser på moduler i sammensæt vare vinduet"</li> </ul>	<ul style="list-style-type: none"> <li>• Vi bør se på om man skal have modulernes priser i systemet, når man vælger moduler.</li> </ul>

Tabel E.10: Kommentarer fra testperson et ved scenarie fire.

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• Testpersonen går ind i special design. "Det er godt med farvekoder, da der er mange ting der skal udfyldes."</li> <li>• Testpersonen søger efter den sofa som kunden vil have.</li> <li>• "Man tilføjer de ønskede moduler som skal på sofaen."</li> </ul>	<ul style="list-style-type: none"> <li>• Testperson 2: "programmet er godt opdelt"</li> </ul>

Tabel E.11: Kommentarer fra testperson to ved scenarie fire.

## Scenarie 5

Du vil oprette et tilbud til en kunde. Dette tilbud består af en specialdesignet vare. Hvordan gør du dette?

Antal deltagere	2
Procent gennemført	100%

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• Testpersonen ville umiddelbart kigge først i ordre info for at løse opgaven.</li> </ul>	<ul style="list-style-type: none"> <li>• Testpersonen besvare opgaven mundtligt i stedet for at bruge mock-ups.</li> </ul>

Tabel E.12: Kommentarer fra testperson et ved scenarie fem.

Kommentarer fra testperson 1	Forslag til forbedringer
<ul style="list-style-type: none"> <li>• "Man kan tilføje varen til ordren"</li> <li>• Testpersonen tror han kan ændre prisen ved at dobbelt klikke på denne</li> <li>• "jeg kan se det er muligt at ændre til og fra tilbud"</li> </ul>	<ul style="list-style-type: none"> <li>• Overvej om funktionaliteten til at dobbeltklikke for at ændre pris skal ind.</li> </ul>

Tabel E.13: Kommentarer fra testperson to ved scenarie fem.

## Afsluttende kommentarer

### Testperson 1

Testperson 1 synes 'Fjern vare' knappen er langt væk. Det er knappen som fjerner en vare fra en åben ordre.

### Testperson 2

- "Det er nemt at se hvilken funktionalitet der hører til hvad"
- "Flot arbejde"
- "Opret ordre skal stå først, hvis man følger Windows standard"



## F | Forandringsstrategi

I dette afsnit vil der beskrives en mulig forandringsstrategi til at facilitere den forandring der må opstå i forbindelse med implementering af et nyt IT-system i Brovst Bolighus. Der tages udgangspunkt i John P. Kotters teorier om forandringsledelse, hvilket står nærmere beskrevet i afsnit A.1.

I det følgende gennemgås der ud fra Kotters 8-trinmodel, hvorledes Brovst Bolighus kan anlægge en strategi til at varetage forandringen af det planlagte IT-system. Forandringsfasen (5., 6. og 7. trin) sættes i relief til den valgte udviklingsmetode Unified Process (UP), for at illustrere hvorledes forandringsprocessen korresponderer med udviklingsprocessen.

**Første trin – Etablér en oplevelse af nødvendighed:** Mads og Brian skal forsøge at motivere medarbejderne til at imødekomme det nye IT-system ved at fremhæve, hvilke klare fordele dette vil medføre. Dette er eksempelvis gevinster i form af øget produktivitet, færre fejl, mere målrettet markedsføring og forøget arbejdsglæde. Derudover henvises til virksomhedens vækststrategi og hvorledes disse gevinster er nødvendige for at imødekomme denne. I den forbindelse fremhæves konkurrenterne, hvor der påpeges at andre bolighuse allerede nyder godt af et IT-system og at IT simpelthen er en naturlig del af fremtiden for at kunne følge med konkurrenterne.

**Anden trin – Oprettelse af en styrende koalition:** Grundet Brovst Bolighus lille størrelse vil koalitionen højst sandsynlig bare bestå af de to ledere Mads og Brian. Det tilstræbes, at den ledende koalition etablerer en tæt kontakt med de øvrige medarbejdere og sørger for at modtage idéer og feedback, således alle føler at have anpart i forandringen.

**Tredje trin – Udvikling af forandringsvision:** Koalitionen udarbejder en vision og strategi for forandringsprocessen, som angiver retning og formål med forandring. Visionen skal afbillede en fremtidig verden, hvor Brovst Bolighus har lyst til at

komme hen. Visionen skal indeholde de første specifikationer og acceptkriterier for det fremtidige IT-system.

Fjerde trin – **Formidling af forandringsvision:** Visionen kommunikeres ud til medarbejderne, hvor der klargøres, hvordan denne vil påvirke og afhjælpe de dagligdagens arbejdsgange. Ydermere tager Mads og Brian udgangspunkt i visionen til formilde den første kravsspecifikation til IT-udviklerne, som også kan bruge denne vision som et pejlemærke.

Femte trin – **Muliggørelse af forandringen:** Når udviklingen af det nye IT-system påbegyndes, begynder virksomheden at anskaffe de nødvendige ressourcer og fjerne eventuelle barrierer for at det nye IT-system kan implementeres. Der kan være tale om anskaffelse af servere, computere m.m. Ydermere bliver medarbejderne allerede på et tidligt stadie sat ind i det nye system via deres inklusion i udviklings inceptionsfase, hvor der laves "tænke højt" tests ved brug af mock-ups og interviews. Her tjenes flere formål, navnlig får udviklerne indsamlet data til udvikling af kravsspecifikation, samt bliver medarbejder allerede her så småt sat i at bruge IT til løse daglige opgaver. Derudover bliver medarbejderne gjort til aktive udøvere af udviklingen og forandringsprocessen, hvilket giver en følelse af medansvar.

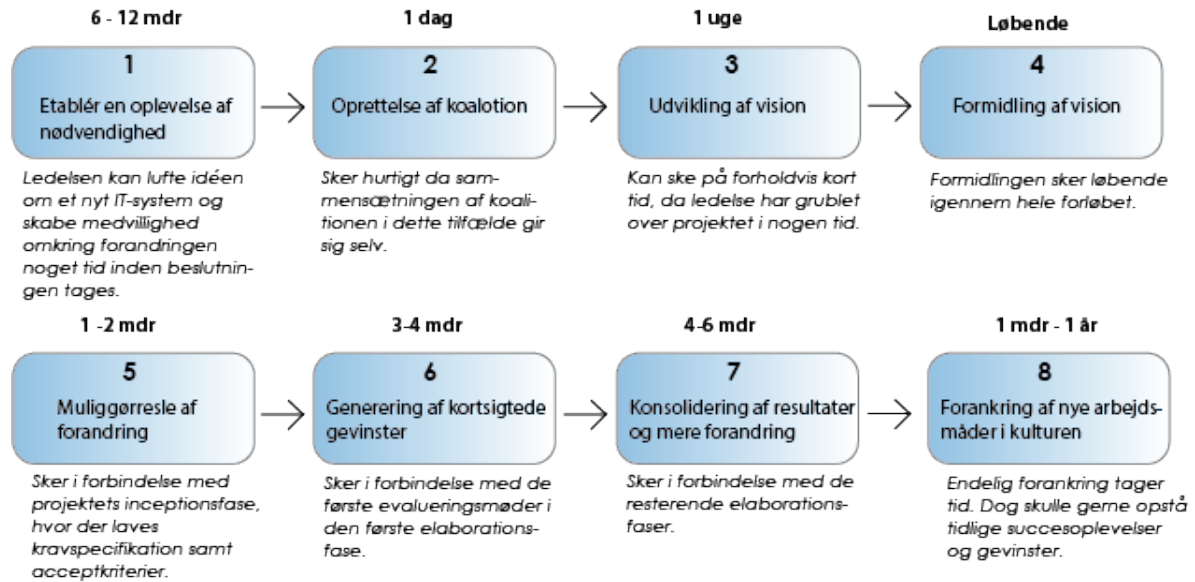
Sjette trin – **Genering af kortsigtede gevinster:** Da udviklerne benytter sig af en iterativ udviklingsmetode, vil der være løbende kontakt og evalueringsmøder med de endelige brugere af systemet. Her vil medarbejder blive præsenteret for funktionelle prototyper som bygger deres egen kravsspecifikation og derved allerede få mulighed for at prøve at løse nogle dagligdagsopgaver ved brug af det nye system.

Syvende trin – **Konsolidering af resultater og produktion af mere forandring:** De kortsigtede succesoplevelser bruges til at skabe grobund for den videre udvikling af IT-systemet, når der påbegyndes nye iterationer indenfor elaborationsfasen. Således vil medarbejdernes positive indstilling anvendes til at arbejde videre med udviklingen af de resterende delsystemer som mangler at blive implementeret.

Ottende trin – **Forankring af nye arbejdsmåder i kulturen:** Når det færdige IT-system er overleveret og tages i brug er der behov for, at de konkrete tiltag får lov til at bundfælde sig. IT som en del af den daglige arbejdsproces skal indlemmes i virksomhedskulturen. Denne ændring i kulturen finder ifølge Kotter først sted, når brugerne bliver overbeviste om at der er tale om at forandringen har medført succes. Derfor skal ledelsen fremhæve og vise tidlige succeser og gevinster på baggrund af det nye system. Nogle af disse gevinster skulle gerne være til at finde på et tidligt stadie i form af forbedret opgaveløsning af f.eks. ordrebehandling og lageroverblik. Andre, som f.eks. øget omsætning og produktivitet vil tage tid at

kunne fremvise.

På figur F.1 vises en tidslinje for Kotters forandringstrin.



**Figur F.1:** Tidslinje for Kotters forandringstrin



# G | Brugerevaluering af SmartOrder

## Usability opgaver

### Opgave 1

Palle Bodilsen henvender sig for at købe en Atlanta-sofa. Du vil derfor gerne bruge systemet til at oprette denne ordre. Du er færdig med opgaven når du har oprettet ordren og er gået tilbage til hovedmenuen.

- Dit brugernavn er "mads" og din kode er "madskode".

### Opgave 2

Bodil Pallesen vil gerne købe en Verona-modulsofa med modulerne 'HCL1' og 'HCL2'. Opret denne ordre. Når du er i hovedmenuen er denne opgave færdig.

### Opgave 3

En kunde kommer ind i butikken og har læst i avisen at Brovst Bolighus er begyndt at udsende nyhedsbreve. Kunden kunne derfor godt tænke sig at modtage dette nyhedsbrev. For at kunne sende nyhedsbrevet til kunden, skal kunden være oprettet i systemet. Kundens navn er Peter Bentson og kundens email er peter@bentson.dk. Resten udfylder du selv.

**Opgave 4**

Bent Pedersen kommer forbi butikken, og ønsker at købe en Albo 2 personers sofa, i eg. Bent Pedersen har ikke handlet ved Brovst Bolighus før. Du skal derfor lave en ordre til Bent Pedersen, med 1 styk Albo sofa.

**Opgave 5**

Du er nu færdig med at bruge programmet, og bedes derfor logge ud.



## H | Systemtests

Test case id	Scenarie/ betingelse	Input				Forventet Resultat
		Vareid	Kunde	antal	discount	
SalesOrder _01	Scenarie 1: Sussesfuld ordre	Atlanta	Bodil Pallesen 5566778 8	2	7%	Varebeskrivelse = "Sofa opstilling 09", Kategori = "Sofa", Deltotal = 1840kr, Salgspris = 1000 kr Adresse = "sidevej 5", Postnr =9400, By= "Noerresundby" Email = "bodil@pallesen.dk Total = 1840kr
SalesOrder _02	Scenarie 2: Ulovligt antal	Atlanta	N/A	-2	N/A	Antal = 0
SalesOrder _03	Scenarie 3: Ulovligt discount	Atlanta	N/A	3	-5% 101%	Discount = 0%
SalesOrder _04	Scenarie 4: Ugyldigt kundenavn eller -telefonnr.	N/A	Bodil Pallesen Eller 5566778 9	N/A	N/A	Popup vindue med teksten "Ugyldigt kundenavn eller telefonnr" Navn, telefon, adresse, postnr, by, email = ""
SalesOrder _05	Scenarie 5: Ugyldigt varenavn eller varenr. ved søgning af specialdesign- varer.	Verona	N/A	N/A	N/A	Popup vindue med teksten "Veronna kan ikke findes i databasen". Model, Leverandør, Beskrivelse, Kategori, = ""; Modulliste = tom
SalesOrder _06	Scenarie 6: Tilføj specialdesign uden valgte moduler.	Verona Module r: Ingen valgt	N/A	N/A	N/A	Popup vindue med teksten "Tilføj venligst moduler". Ordrelinje-tabel forbliver tom.

Figur H.1: Systemtests med data



Test case id	Scenarie/ betingelse	Input				Forventet Resultat
		Vareid	Kunde	antal	discount	
SalesOrder _01	Scenarie 1: Sussesfuld ordre	V	V	V	V	<ul style="list-style-type: none"> <li>Varebeskrivelse, varekategori og pris svarer til valgte vare.</li> <li>Telefonnr, adresse, email, svarer til søgte kundenavn.</li> <li>Deltotal på ordrelinje passer med varens pris gange antal minus den angivne discountsats.</li> <li>Total stemmer med summen af subtotaler.</li> </ul>
SalesOrder _02	Scenarie 2: Ulovligt antal	V	N/A	I	N/A	Hvis antal er et negativ tal sættes antal til 0 og der gives en fejlmedling til bruger.
SalesOrder _03	Scenarie 3: Ulovligt discount	V	N/A	V	I	Hvis discountsatsen underskrider 0% eller overskrider 100% sættes discount tilbage til 0 og bruger modtager en fejlmedling.
SalesOrder _04	Scenarie 4: Ugyldigt kundenavn eller -telefonnr.	N/A	I	N/A	N/A	Trykkes Enter-tasten og det indtastede søgedata (navn eller telefon) ikke matcher nogle kunder i databasen modtager bruger en fejlmedling.
SalesOrder _05	Scenarie 5: Ugyldigt varenavn eller varenr. ved søgning af specialdesign-varer.	I	N/A	N/A	N/A	Trykkes Enter-tasten og det indtastede søgedata (varenr eller varenavn) ikke matcher nogle varer i databasen modtager bruger en fejlmedling.
SalesOrder _06	Scenarie 6: Tilføj specialdesign uden valgte moduler.	I Modu ler: I	N/A	N/A	N/A	Tilføjes en specialdesignet vare uden nogle valgte moduler modtager bruger en fejlmedling.
SalesOrder _07	Scenarie 7: Annullere ordre	V	V	V	V	Bruger ønsker at annullere hele ordren. Bruger bedes bekræfte og derefter slettes ordren.
SalesOrder _08	Scenarie 8: Gem ordre som tilbud.	V	V	V	V	Brugeren bedes indtaste en forfaldsdato på tilbuddet. Ordrestatus sættes til tilbud.

Figur H.2: Systemtests uden data



# I | Brugervejledning

For at starte programmet køres klassen LoginView. Du bliver bedt om at logge ind, hvor følgende logins kan bruges:

Username: mads

Password: madskode

Username: brian

Password: brianskode

Efter login er det muligt at vælge mellem tre menuer.

Opret salgsordre, produkter, og kunder, ansatte og leverandører.

**Opret salgsordre** For at oprette en salgsordre trykkes der på knappen 'Opret salgsordre'. Der går et øjeblik, da data skal hentes fra databasen.

I det nye vindue kan du se alle informationer der skal bruges for at oprette en salgsordre.

Under Ordreinfo i øverste venstre hjørne kan du se om der er forbindelse til database og dato. Hvis der står 'Online' i grøn skrift, er der forbindelse til databasen, og du kan gå igang.

For at oprette en salgsordre, skal du vælge fra 'Tilføj vare' panelet i højre side. Her er der to muligheder, enten kan du vælge en hyldevare eller specieldesign.

Hvis det ønskes at søge gennem hyldevare, er det muligt at bruge 'Filtrer vare'. Hvis feltet er tomt vises samtlige vare. Ellers kan der søges på f.eks. 'Atlanta'. Efter ønsket vare er fundet bliver den vist i vinduet liger under søgefeltet. Den ønskede vare markeres og der indtastes ønsket antal i 'Antal' feltet i nederste højre hjørne. Herefter trykkes der på 'Tilføj vare'.

For at vælge specieldesign trykkes der på fanen 'Specialdesign'. For at søge efter

en model benyttes 'Søg model' feltet. Der kan søges på bogstaver som modelnavnet indeholder, eller selve modelnavnet. Hvis den ønskede model findes, trykkes der på den, og 'Model', 'Leverandør', 'Beskrivelse' og 'Kategori' felterne udfyldes automatisk. Herefter er det muligt at vælge moduler til basismodellen fra vinduet i nederst venstre hjørne. Hvis det ønskes kun at se bestemte modultyper, kan dette vælges fra dropdown menuen 'Modultype'. For at vælge et modul markeres dette, og der trykkes på den grønne 'tilføj' knap. Det valgte modul vises i højre side under 'Valgte moduler'. Disse kan fjernes igen ved at bruge den røde 'Fjern' knap. Når bestillingen er færdig bruges den gule 'Tilføj' knap.

Når alle ønskede vare er tilføjet, kan der tilføjes en kunde til ordren. Dette gøres på samme måde i midten af venstre side. Der søges i feltet 'Søg kunde via navn' med navn på kunden. F.eks. 'Bodil', eller 'Palle'. Kunden markeres og indformationen bliver automatisk udfyldt.

I nederste del af vinduet ses 'Ordrelinjer', som viser alle vare tilføjet til den nuværende ordre. Ønskes en vare fjernet markeres den, og der trykkes på 'Fjern vare fra ordre'.

Når kunden er tilføjet og det ønskede antal vare er tilføjet kan der trykkes på 'Opret tilbud' for at oprette og gemme et tidsbegrænset tilbud. Benyt 'Opret ordre' for at gemme ordren i databasen med det samme. 'Annuller' bruge for at slette alle indtastede data og vende tilbage til hovedmenuen.

**Produkter** Trykkes der på Produkter knappen åbnes en undermenu hvor det kan vælges "Opret Produkt" eller "Opret Produkttype"

I Opret Produkt er det muligt at tilføje nye hyldevare, specialdesign eller moduler.

Dette gøres ved at vælge fra dropdown menuen under 'Vælg produkttype'. Når der er valgt type, indtastes de nødvendige indformationer og der vælges leverandør fra dropdown menuen under 'Leverandør'. Hvis det er et modul til en bygselv sofa vælges der også til hvilken sofa modulet hører til, under 'Modul til' dropdown.

Når alle informationer er tastet korrekt oprettes varen ved at trykke på 'Opret vare' knappen. Varen eller modulet ligges derefter i databasen. Denne kan hentes op igen fra 'Opret salgsordre' igen.

**Kunder, ansatte og Leverandører** Hvis det ønskes at oprette eller administrere personer i systemet kan dette gøres gennem kunder, ansatte og leverandør knappen.

Der åbnes et nyt vindue, hvor der kan vælges hvilken type der ønskes at oprette

eller administrere.

**Opret kunde** For at oprette en kunde trykkes der på 'Opret kunde' Kundens informationer indtastes i relevante felter, og der vælges hvilken type kunde der tales om, privat eller erhverv. For at gemme kunden i databasen trykkes på 'Tilføj kunde'. Hvis det ikke ønskes at gemme kunden bruges knappen 'Annulere' for at slette indtastet information. 'Tilbage' knappen bruges for at vende tilbage til tidligere menu.

Opret ansat og leverandør, administrer kunder, ansatte og leverandører er ikke implementeret.



## J | Create scripts til databasen

```
1 use dmab0916_202650;
2
3 create table ZipCity (
4     zip_code varchar(8) primary key,
5     city varchar(40) not null
6 )
7
8 create table Person (
9     id int primary key identity(1,1),
10    name varchar(40) not null,
11    address varchar(40) not null,
12    zip_code varchar(8) foreign key references
13        ZipCity(zip_code) on update cascade on delete cascade,
14    email varchar(40),
15    person_type varchar(8)
16 )
17
18 create table Phone (
19     phone varchar(20) primary key,
20     person_id int foreign key references Person(id) on
21         update cascade on delete cascade
22 )
23
24 create table Employee (
25     id int primary key foreign key references Person(id) on
26         update cascade on delete cascade,
27     ssn varchar(20),
28     salary float,
29     username varchar(20)
```

```
27 )
28
29 create table Password (
30     id int foreign key references Employee(id) on update
31         cascade on delete cascade,
32     password varchar(20)
33 )
34
35 create table Customer (
36     id int primary key foreign key references Person(id) on
37         update cascade on delete cascade,
38     type varchar(20)
39 )
40
41 create table Supplier (
42     id int primary key foreign key references Person(id) on
43         update cascade on delete cascade,
44     cvr varchar(20)
45 )
46
47 create table OrderCondition (
48     id int primary key identity(1,1),
49     type varchar(20)
50 )
51
52 create table OfferType (
53     id int primary key foreign key references
54         OrderCondition(id) on update cascade on delete cascade,
55     create_date date,
56     due_date date,
57     accept_date date
58 )
59
60 create table OrderType (
61     id int primary key foreign key references
62         OrderCondition(id) on update cascade on delete cascade,
63     create_date date,
64     pack_date date
65 )
```



```

61
62 create table DeliveredType (
63     id int primary key foreign key references
64         OrderCondition(id) on update cascade on delete cascade,
65     date date
66 )
67
68 create table ProductPrice (
69     id int primary key identity(1,1),
70     purchase_price float,
71     sales_price float,
72     from_date date
73 )
74
75 create table ProductType (
76     id int primary key identity(1,1),
77     category_name varchar(40),
78     type varchar(20),
79 )
80
81 create table Product (
82     id int primary key identity(1,1),
83     model varchar(100),
84     description varchar(1000),
85     dimensions varchar(40),
86     product_price_id int foreign key references
87         ProductPrice(id) on update cascade on delete cascade,
88     product_type_id int foreign key references
89         ProductType(id) on update cascade on delete cascade,
90     supplier_id int foreign key (supplier_id) references
91         Supplier(id)
92 )
93
94 ALTER TABLE ProductType ADD template_id int foreign key
95     references Product(id);
96
97 create table PartOfProduct (
98     id int primary key identity(1,1),
99     product_part_id int foreign key references Product(id),

```

```
95     product_id int foreign key references Product(id) on
96         update cascade on delete cascade
97 )
98 create table Property (
99     id int primary key identity(1,1),
100     name varchar(40),
101     string_value varchar(40),
102     double_value int,
103     boolean_value bit,
104     type varchar(20),
105     product_id int foreign key references Product(id)
106 )
107
108 create table SalesOrder (
109     id int primary key identity(1,1),
110     date_placed date,
111     paid bit not null default(0),
112     order_sent bit not null default(0),
113     employee_id int foreign key references Employee(id),
114     customer_id int foreign key references Customer(id) on
115         update cascade on delete cascade,
116     order_condition_id int foreign key references
117         OrderCondition(id) on update cascade on delete cascade
118 )
119
120 create table SalesOrderLine (
121     id int primary key identity(1,1),
122     order_id int foreign key references SalesOrder(id) on
123         update cascade on delete cascade,
124     product_id int foreign key references Product(id) on
125         update cascade on delete cascade,
126     amount int,
127 )
128
129 create table SupplyOrder (
130     id int primary key identity(1,1),
131     date date,
```

```
128     supplier_id int foreign key references Supplier(id) on
        update cascade on delete cascade,
129 )
130
131 create table SupplyOrderLine (
132     id int primary key identity(1,1),
133     supply_order_id int foreign key references
        SupplyOrder(id) on update cascade on delete cascade,
134     product_id int foreign key references Product(id) on
        update cascade on delete cascade,
135     amount int,
136     product_price_id int foreign key references
        ProductPrice(id)
137 )
```