Datavetenskap

Therese Sundström

Utveckling av ett affärssystem med Unified Process

Examensarbete, D-nivå 30 ECTS

2005:05

Utveckling av ett affärssystem med Unified Process

Therese Sundström

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en civilingenjörsexamen i informationsteknologi. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Therese Sundström

Godkänd, 2005-06-07

Opponent: Per Johansson

Opponent: Henrik Wallinder

Handledare: Katarina Asplund

Examinator: Donald F.Ross

Sammanfattning

Denna uppsats beskriver ett projekt utfört inom ramen för Unified Process (UP). Unified Process är en Software Engineering metod som innehåller hela utvecklingsprocessen från kravinsamling till testfall och överlämning. Uppsatsen ger också en inblick i området Software Engineering samt dess historia. Projektet som utförs är på uppdrag från företaget Ponderator AB i Karlstad. Företaget önskade ett affärssystem för hantering av offerter, order, fakturor och produkter. Även ett kundregister och möjligheten att lätt kunna ändra kunddata behövdes. Dessutom ville företaget också kunna koppla sina offerter, order och fakturor till dokument som kunde skrivas ut och skickas till kunden.

Projektet utfördes i nära samarbete med kunden och systemet utvecklades helt efter företagets önskemål. Systemet innehåller en databas skapad i MySql samt ett grafiskt användargränssnitt implementerat i Java. Systemet är nu i drift och håller på att utvärderas av användarna. De olika stegen i Unified Process beskrivs i samband med resultatet av varje steg. UP visade sig vara en bra metod även för en enskild utvecklare, eftersom de delar av processen som är mest relevanta för projektet kan plockas ut.

Development of a Business System with Unified Process

Abstract

This thesis describes the development of a business system with the support of the Software Engineering process Unified Process. Unified Process is a Software Engineering methodology and it contains models for all of the major steps in the Software Engineering area. The steps goes from requirement capture to test cases and deployment. The thesis also gives an overview of the Software Engineering area and its history. The development of the system was an assignment from the company Ponderator AB in Karlstad. The company had a need for a system that could help them handle offers, orders, invoices as well as product and customer information. In addition the company wanted to be able to transfer their offers, orders and invoices to printable documents.

The project was performed in close cooperation with the users of the system and it was developed entirely from the company's requirements. The system consists of a relational database created in MySql and a graphical user interface implemented in Java. The system is installed and is now being evaluated by the users. Every step of the development process has been documented and described along with the result. The Unified Process was a good choice even for a single developer since the most relevant parts of the process can be used for the project specific needs.

Innehållsförteckning

Figurförteckning

Tabellförteckning

1 Inledning

Denna uppsats att ger en inblick i hur Unified Process kan användas i praktiken. I uppsatsen kommer jag att beskriva bakgrunden till Software Engineering, ge exempel på metoder som används och sedan utföra ett projekt med hjälp av en av metoderna.
Det projekt jag ska utföra är att skapa ett nytt affärssystem åt företaget Ponderator AB. Arbetet kommer att utföras enligt metoden Unified Process (UP), vilket finns beskrivet i avsnitt 2.6. UP riktar sig egentligen mot hela utvecklingsteam, men jag kommer att följa processen så långt som möjligt.

Ponderator AB är ett nystartat företag i Karlstad. Företaget utför lasergravyr på olika typer av föremål, så som glas och namnbrickor. Företaget riktar sig främst till andra företag men även i viss mån till privatpersoner. Jag fick i uppgift att skapa ett helt nytt affärssystem åt företaget. Systemet skulle innehålla bland annat en databas med kunder, produkter, offerter, order och fakturor. Systemet krävde också ett gränssnitt för att lägga till, ta bort och redigera information om kunder och övriga entiteter. Företaget ville också ha hjälp med att skapa en mall för offert, order och faktura i MS Word eller liknande. Utvecklingen skedde med hjälp av UP och de modeller som finns inom UP. Enligt Software Engineering-principen arbetade jag i samarbete med användarna för att ta fram de krav som fanns på systemet.

Projektet resulterade i en mySql databas som lagrar information om offerter, order, fakturor, produkter och kunder. För att kunna skapa de ovan nämnda objekten har också ett grafiskt gränssnitt mot databasen skapats. Gränssnittet är implementerat i Java och är uppbyggt enligt Model-View-Controller modellen.
De mallar som företaget ville ha skapades i MS Excel. Mallarna har automatisk uträkning av summor med och utan moms. Användaren kan också enkelt föra in produkter från en faktura i databasen genom MS Query.

1.1 Disposition

Uppsatsen börjar med en inledning till Software Engineering i kapitel 2. Kapitlet innehåller historia, en beskrivning av processen som används inom SE, samt ett antal modeller och metoder. Efter beskrivningen av området börjar kravinsamlingen och en beskrivning av användningsfallsmodellen i kapitel 3. Kapitlet innehåller specifikation av funktionella och ickefunktionella krav samt de användningsfall som skapats utefter kraven. Kravinsamlingen övergår sedan i kravanalys i kapitel 4, där en första blick av systemets arkitektur ges genom ett antal samarbetsdiagram.

Efter analysen av kraven skapas systemets design. I kapitel 5 beskrivs designmodellen och databasen. I detta kapitel finns även en beskrivning av den designprincip som använts. Kapitel 6 beskriver implementationen av den design som beskrivs i kapitlet före. Här finns en beskrivning av det grafiska användargränssnittet samt av funktionaliteten som ligger i modellen och kontrollen. I nästkommande kapitel 7 beskrivs ett antal testfall som skapats för att testa det färdiga systemet. Kapitlet beskriver också testmodellen som ingår i UP.
Kapitel 8 sammanfattar resultatet av uppsatsen och ger ett antal rekommendationer för framtida arbete med systemet. Kapitlet innehåller även uppsatsens slutsats.

2 Bakgrund

Jag kommer i detta kapitel att ta upp olika aspekter inom området Software Engineering (SE). I avsnitt 2.1-2.3 kommer jag att beskriva bakgrunden till SE och ge en överblick av området. I avsnitten 2.4, 2.5 och 2.6 kommer jag att beskriva tre olika metoder som används inom SE. Den sista av dessa metoder (Unified Process) ska jag sedan applicera under mitt praktiska arbete på Ponderator AB.

2.1 Software Engineering

Software Engineering är idag ett ganska känt begrepp till namnet. Däremot är inte alltid bakgrunden och behovet av SE lika känt. För att förklara behovet av SE får man gå tillbaka till den tid då programutvecklingen började ta fart.

2.1.1 Historia

I programmeringens början, under 1950-talet, var problemen som skulle lösas väl förstådda [7]. I många fall var den som ville ha ett dataprogram densamma som skrev det. Detta berodde mycket på att datorer var stora och komplexa och inte minst mycket dyra. Eftersom användaren var den samma som utvecklaren skedde ingen feltolkning mellan vad som önskades och vad som skapades. Under 1960-talet blev datorerna billigare och mindre. Datorer började därigenom användas av fler och fler samt inom fler områden. Problemen som uppstod var nu mer komplexa än tidigare. Större och mer specialiserade program krävdes, samtidigt som de skulle vara enklare att förstå och använda.

Det var under 1960-talet programmering blev till ett yrke. Vanliga datoranvändare kunde anlita programmerare för att lösa de problem som uppstod. Det var också nu bristen på kommunikation och förståelse mellan människor inom olika områden började bli ett problem. Den som var i behov av programmet var kanske en mycket lärd kemist som ville ha ett program som räknade ut svåra matematiska formler. Programmeraren å andra sidan hade mycket bra datakunskaper men visste inget om kemi. Problemet var att för kemisten förklara exakt vad han/hon ville ha, samt för programmeraren att förstå detta och göra en väl anpassad implementation.

Det var i mitten på 1960-talet som det först blev aktuellt att skapa stora mjukvarusystem. Det var också under den tiden som de första och bäst dokumenterade operativsystemen skapades. När dessa stora mjukvarusystem skulle skapas uppstod naturligtvis problem. Det fanns ingen tidigare erfarenhet av att arbeta med, och utveckla, så stora system. Det visade sig inte alls vara samma sak att utveckla ett applikationsprogram som att skapa ett operativsystem.

Arbetet bestod nu mest av att kommunicera med andra som deltog i samma arbete. Det var inte bara att utföra en rad maskininstruktioner längre. Här uppstod nu utrymme för tolkning av kraven på mjukvaran. Trots att man vid denna tid insåg komplexiteten i att utveckla stora system så blev produkter sällan färdiga i tid. I de flesta fall drog de även över kostnadsbudgeten. Detta ledde till att uttrycket ”software crisis” myntades. Det fanns inga exakta metoder för att utveckla komplexa system. Det var också svårt att ge några garantier. För att lösa detta problem så utgick ingenjörerna från att mjukvarusystemet skulle ses som en stor slutprodukt. Denna slutprodukt skulle skapas på samma sätt som ingenjörer inom andra områden skapade sina slutprodukter.
Det är ur denna föreställning begreppet Software Engineering utvecklades till vad det är idag. En första modell för hur mjukvaruutvecklingsprocessen skulle gå till arbetades fram, den kom att kallas vattenfallsmodellen. Denna modell blev startpunkten för utvecklingen av flera olika typer av modeller (se avsnitt 2.2 och 2.3).

2.1.2 Översikt av SE

Software Engineering kan enligt [7] ses som ”en applicering av ingenjörskonst på mjukvara”. Området utvecklades för att dela in stora komplexa projekt i mindre delar och på fler personer. Idag skapas system för att sedan finnas i flera olika versioner. Samtidigt ändras vem som gör de olika versionerna kontinuerligt. Något som då är viktigt är bra dokumentation och en bra kommunikation med kunden.
I många fall är inte beställaren av mjukvaran den samma som användaren. Detta kan vara ett stort problem då användaren är oerhört viktig i utvecklingen. Beställaren har ofta systemkrav och restriktioner, men inte några direkta anspråk på användbarheten. SE erbjuder en modell för att se till att alla krav blir tillgodosedda genom att tillsammans med kunden (vare sig det är beställare eller användare) utarbeta en kravspecifikation.

Precis som inom många andra områden inom datavetenskap finns det begrepp inom SE som är viktiga att veta definitionen på. Nedan följer en beskrivning av några av de begrepp som kommer att användas i uppsatsen [16].

- **Process**

  En process inom SE kan ses som en mängd aktiviteter som resulterar i en färdig produkt. Processen inkluderar både utvecklingen och resultatet. Processen bestämmer i vilken ordning som stegen inom utvecklingen ska tas. Stegen inom SE beskrivs vidare i avsnitt 2.2
- **Modell**

  En modell är en förenklad beskrivning av en process. Modellen visar en mer abstrakt vy av processen och beskriver den ur en särskild synvinkel. Ett antal modeller beskrivs närmare i avsnitt 2.3
- **Metod**

  En metod är ett strukturerat sätt att utföra mjukvaruutveckling. Det finns ingen metod som är så generell att den passar alla projekt, istället finns metoder som är anpassade till olika typer av projekt. Några exempel är objektorienterade och funktionsorienterade metoder. Metoder beskrivs vidare i avsnitten 2.4, 2.5 och 2.6
- **Produkt**

  En produkt behöver inte innebära slutprodukten eller det färdiga systemet. En produkt kan också vara en prototyp eller ett dokument från ett steg i processen
- **Kund**

  Kunden är oftast den som beställer tjänsterna från en mjukvaruutvecklare. Kunden är inte alltid densamma som användaren av systemet
- **Användare**

  Personen som faktiskt ska använda slutprodukten. Denna person är mycket viktig i utvecklingen

2.2 Översikt av utvecklingsprocessen

I detta avsnitt ges en överblick av de steg som ingår i de flesta utvecklingsprocesser och metoder. Utseendet kan variera mellan processerna/metoderna men de följande stegen är grundläggande och ingår därför i de flesta av dem. Det är en generell beskrivning och därför kommer jag inte att ta upp så mycket detaljer. Senare i kapitlet kommer jag att presentera olika metoder och därigenom också mer detaljer. Avsnitten börjar med en överblick av kravinsamling och kravanalys och fortsätter vidare med design och implementation, testning och underhåll.

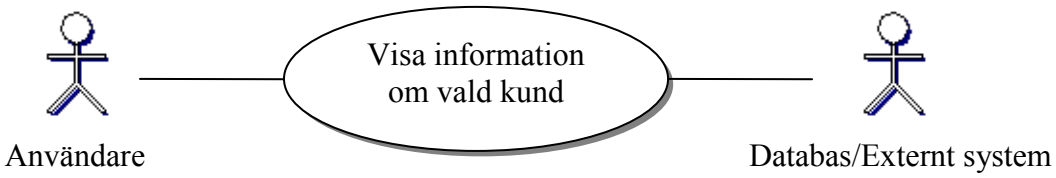
Den inledande undersökningen som görs när ett företag eller en kund vill ha ett nytt datasystem eller utöka ett befintligt system är en så kallad *feasibility study* [9]. Med feasibility study menas att någon av de utvecklare som ska arbeta med projektet tar reda på om det är genomförbart och om det är kostnadseffektivt. Denna studie ligger till grund för om arbetet ska fortsätta eller inte.

2.2.1 Kravinsamling och kravanalys

Det första steget i utvecklingen är att samla in krav (*requirement capture, requirement analysis*). Krav innebär vad slutprodukten ska utföra och hur den ska se ut. Kravinsamlingen och analysen av kraven är en stor del i ett SE-projekt. Anledningen till att kravinsamling och analys sker i så stor utsträckning är att en stor del av produktens kostnad tidigare har gått till krav som förändrades under utvecklingsprocessens gång. Att dessa förändringar sker betyder ofta att utvecklaren och kunden/användaren inte har samma uppfattning om vad produkten ska utföra. Att kraven förändras är mer regel än undantag i mjukvarubranschen. Skillnaden mellan att ha missförstått kraven från början och att kraven och förutsättningarna förändras är stor. SE ofta en iterativ process där kraven kan förändras och förfinas under processens gång. Genom att strukturera och analysera kraven kan ett projekt få mycket bättre resultat [17].

Det finns olika typer av krav: funktionella krav samt ickefunktionella krav. Kraven kan komma från användaren eller från ett befintligt system. De funktionella kraven omfattar hur systemet ska fungera, vad en viss inparameter ska ge för resultat och ibland även vad systemet inte ska göra [12]. Icke-funktionella krav handlar om vilka begränsningar som finns, tidskrav och krav på standard. Användarens krav ska hanteras på ett sådant sätt att användaren själv kan förstå dem. Kraven ska vara tydliga och skrivna på ett språk som användaren kan förstå.

Under specifikationen av krav ska inga interna delar av systemet specificeras utan endast de yttre. Systemkraven är en mer detaljerad version av användarkraven och ska spegla hela systemet väl. Systemkraven kan då ligga till grund för hur implementationen kommer att se ut. Under insamlingen av systemkrav används olika modeller av systemet. Modellerna visar olika vyer av systemet, så som objekt-vy och informationsflödet i systemet. Det är under denna fas man tar hjälp av olika typer av modellspråk, det mest använda är UML (Unified Modeling Language) [14]. För att illustrera användarens krav och funktionella krav används ofta användningsfallsdiagram. Dessa diagram innehåller en eller flera aktörer som är intressanta för systemet. Aktören, till exempel en slutanvändare, är källan till en händelse, till exempel att hämta information om ett objekt. Objektet kan vara en kund, en beställning eller liknande. Informationen finns oftast hos en annan aktör av något slag, det kan vara ett externt system eller en databas. Diagrammet kan till exempel se ut på följande vis:



Figur 2.1: Exempel på användningsfallsdiagram

Kravanalysen resulterar i ett eller flera dokument (*requirement document*), dessa dokument ska innehålla kraven för den framtida utvecklingen.
Nedan följer några viktiga punkter som dokumentet ska innehålla:

1. Slutproduktens betydelse – varför ska den utvecklas, vad den har för fördelar
2. Vilka personer som har intresse i produkten
3. Vilka personer som har användning av produkten
4. Krav och begränsningar på produkten
5. Funktionella krav
6. Ickefunktionella krav
7. Antaganden som har gjorts vid kravinsamlingen
8. Slutproduktens kopplingar till andra delar av systemet
9. Slutproduktens kostnad

Under punkt två och tre ovan kommer begreppet intressenter (*stakeholders*) in. Detta är ett viktigt begrepp inom SE, eftersom kraven kan se olika ut beroende på vilken intressent man pratar med. Kraven samlas in från intressenter med olika infallsvinklar eftersom intressenterna ser på produkten från olika håll. Versionshantering av det skapade kravdokumentet är mycket viktigt, dokumentet kan komma att ändras flera gånger under utvecklingen av mjukvaran.

2.2.2 Design och implementation

Designen är det första steget som för kraven och specifikationen av produkten mot ett exekverbart system och den fungerar som en beskrivning av systemets struktur. Syftet med designen är att få en djupare förståelse för ickefunktionella krav, samt att dela upp den kommande implementationen i mindre delar. Designen hjälper även till att hitta viktiga gränssnitt i systemet så att de kan definieras väl. Beskrivningen av systemet görs i olika detaljnivåer beroende på vad det är som ska specificeras, eftersom om hela systemet skulle designas på specifik nivå med en gång skulle det bli mycket ostrukturerat. Ofta börjar designfasen med att skapa en struktur på hur systemets stora delar ska fungera ihop, kanske med en klient-server modell. Sedan bryts systemet ner i delar så att varje del kan struktureras mer i detalj. Nedbrytningen gör att strukturen blir mer översiktlig eftersom det inte blir så mycket i varje del för sig. Designfasen använder sig av olika typer av modeller beroende på vad det är som ska specificeras.
Olika typer av modeller som används är:

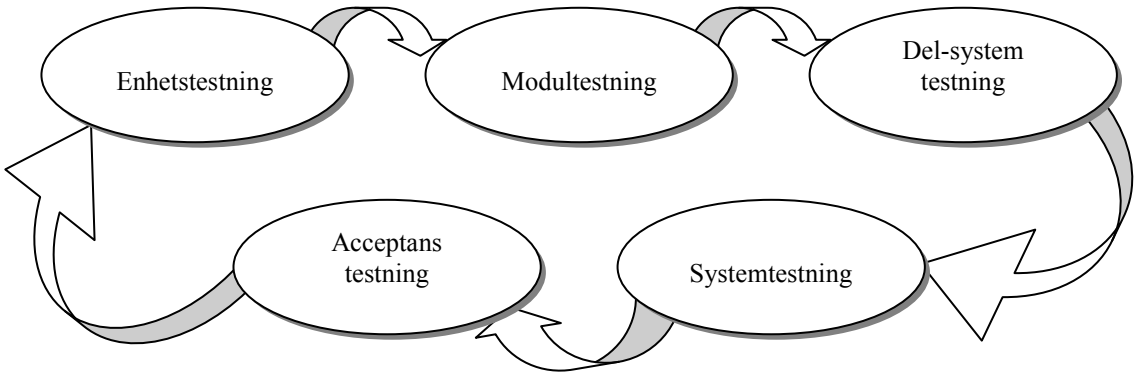
- Dataflödesmodellen – Visar hur information och data flödar genom systemet och hur den förändras under flödet
- Entitets/relationsmodellen – I denna modell tas de grundläggande entiteterna upp, modellen visar också samspelet mellan entiteterna
- Modell över systemets struktur – Visar systemets komponenter och samarbetet mellan dem
- Objektorienterade modeller – Visar hur systemet ser ut i objektorienterad form, med arv, statiska och dynamiska aspekter, kan vara uppdelad på flera modeller

Målet med designfasen är att hitta strukturen för, och beskriva systemet, på ett sådant sätt att det blir en smidig övergång mot implementationen. Designen ger en strukturerad väg att gå till skillnad från att använda en ad-hoc liknande process som oftast tar längre tid i och med att många fel och brister upptäcks för sent i utvecklingsprocessen.

I implementationsfasen realiseras det designade systemet. Det är här realisering av det designade systemet sker. Implementationsfasen ligger ganska sent i utvecklingsprocessen (beroende av process), och det är just detta som förvånar många då de kommer i kontakt med SE första gången. Meningen är att implementationen inte ska påbörjas förrän systemet är väl förstått och arkitekturen för systemet ska vara så tydlig att övergången mot implementationen blir så gott som sömlös. Precis som de tidigare beskrivna faserna sker implementationen iterativt. Denna fas ingår i livscykeln för produkten och byggs på steg för steg.

2.2.3 Testning

Testning är en mycket viktig fas, den görs iterativt och initialt på små delar av systemet. Testningsfasen kallas också för systemets valideringsfas. De första delarna behöver inte vara färdiga komponenter i systemet utan kan vara ett litet antal rader kod. Testningsfasen kan illustreras i Figur 2.2:



Figur 2.2: Testningsfasen

Dessa faser beskrivs på följande sätt i [16]:

1. Enhetstestning – Varje individuell komponent testas utan att kopplas till andra komponenter i systemet
2. Modultestning – En modul kan beskrivas som en samling av oberoende komponenter. Modulen sammankopplar komponenter som har en gemensam uppgift. Dessa testas sedan gemensamt
3. Delsystemtestning – Testar samlingar av moduler som formar en del av systemet. Ett vanligt problem är att gränssnitt inte matchar varandra. Detta bör därför uppmärksammas under denna del av fasen
4. Systemtestning - Testar hela systemet. Målet är att hitta de fel som uppstår mellan subsystemen som nu har satts ihop. I denna del-fas sker även validering av de ickefunktionella och de funktionella kraven som ställts
5. Acceptanstestning - Systemet testas nu i sin rätta miljö. Systemet får rätt form av data från kunden och användaren

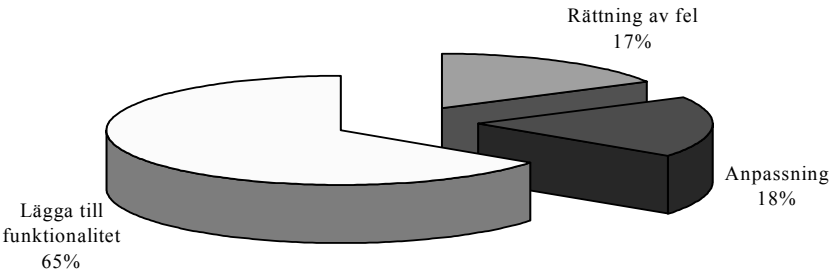
De första två stegen i testningsfasen görs oftast av dem som skriver koden för just den enheten. Testningen utförs inkrementellt samtidigt som enheten växer. Testfasen innebär att testfall skapas för att kunna testa varje enhet för sig. Testfallen skrivs ofta före implementationen börjar, detta är också lite beroende av vilken metod som används. (se avsnitt 2.4 – 2.6.

2.2.4 Underhåll

Underhåll (*maintenance*) handlar om att förändra eller rätta till fel i systemet efter att det har levererats. Största delen av ett företags kostnader för dess datasystem ligger under underhåll (ungefär 60 procent) [16]. Det finns inte någon metod eller process som kan skapa ett datasystem som inte måste gå igenom någon typ av förändring. Förändringar kan vara av olika typ. Systemet kan lida av programmeringsfel samtidigt som det kan uppstå nya krav på systemet. Det är mycket svårt att förutse underhåll och därför är det också en så stor kostnad. Olika typer av underhåll är:

1. Att reparera fel i systemet (*Corrective maintenance*)
2. Att anpassa systemet till en ny miljö (*Adaptive maintenance*)
3. Att lägga till eller förändra systemets funktionalitet (*Perfective maintenance*)

Dessa olika typer av underhåll fördelar sig på följande sätt [16]:

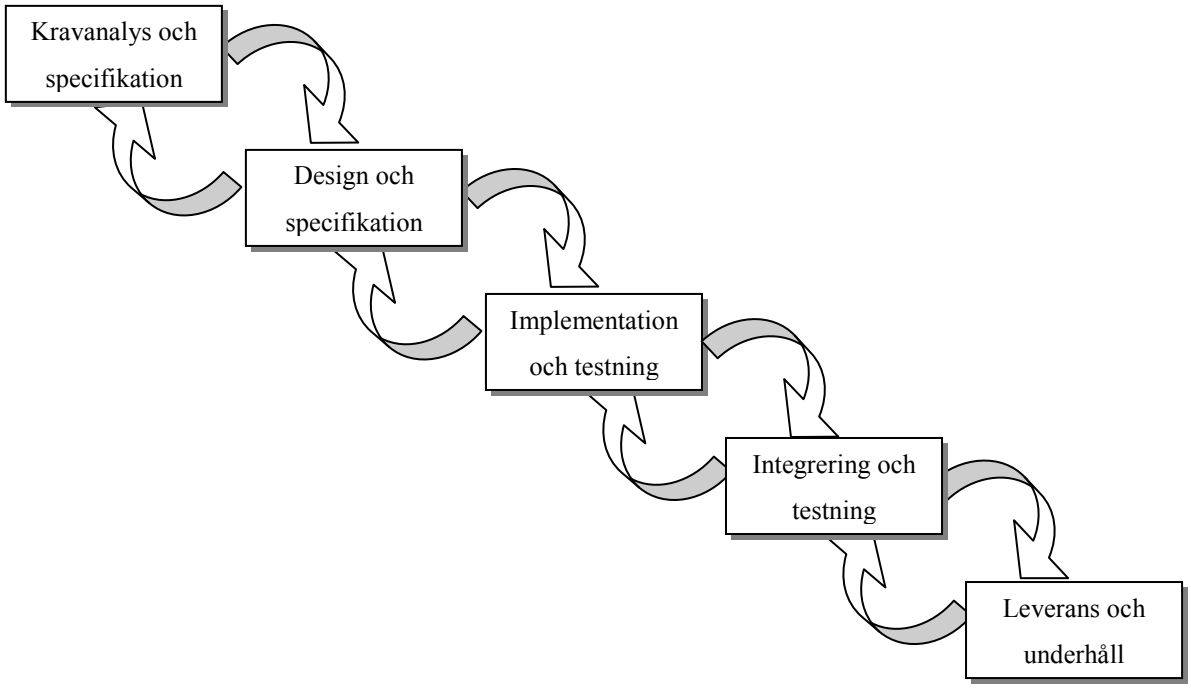


Figur 2.3: Kostnadsfördelning för underhåll

2.3 Modeller och processer inom Software Engineering

2.3.1 Vattenfallsmodellen

Inom SE används termen *livscykel* när utvecklingen av mjukvara diskuteras [15]. Denna term betyder att mjukvaran utvecklas stegvis och genom olika faser. För varje fas byggs produkten på för att resultera i en slutprodukt. Detta beskrevs som en produkts livscykel i vattenfallsmodellen [16] (se Figur 2.4) som var den första modellen som anammades inom SE.

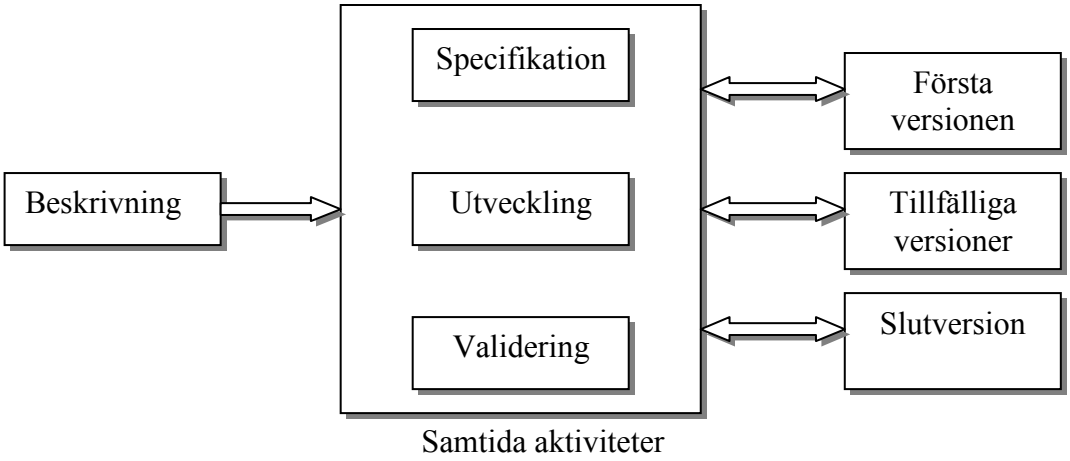


Figur 2.4: Vattenfallsmodellen

Denna modell antyder att varje fas avslutas innan nästa påbörjas. Så är dock sällan fallet i verkligheten. Den första versionen av vattenfallsmodellen antyder även att varje fas utförs endast en gång (nedåtgående pilar). Modellen vidareutvecklades till att innehålla pilar även åt motsatt håll (uppåtgående pilar). Det vill säga, det skapades en cykel i utvecklingen, genom att gå tillbaka till starten och samla in nya krav och gå vidare framåt igen. Vattenfallsmodellen är dokumentorienterad, vilket betyder att varje fas resulterar i ett dokument av något slag. Dokumentet ligger sedan till grund för nästa fas i processen. Problemet är att dokumentet från föregående fas ofta inte innehåller tillräckligt med information. Faserna behöver överlappa varandra för att tillräcklig information ska finnas.

2.3.2 Evolutionär utveckling

Modellen evolutionär utveckling (*evolutionary development*) (se Figur 2.5) separerar inte de olika faserna lika tydligt som vattenfallsmodellen (se Avsnitt 2.3.1) och spiralmodellen (se avsnitt 2.3.3). I denna modell överlappar specifikation, utveckling och verifiering varandra. Detta innebär att det snabbt skapas en produkt som kunden kan ge feedback på [13]. Denna modell delas ofta in i två kategorier, *exploratory development* och *throw-away prototyping*. Den förstnämnda används då en kravspecifikation är mycket svår att skapa från början. Istället utforskas systemet och byggs på bit för bit i nära samarbete med kunden och/eller användaren. Den senare modellen används då kundens krav är svåra att specificera. Det innebär att en prototyp skapas i varje iteration som sedan utvärderas och slängs.



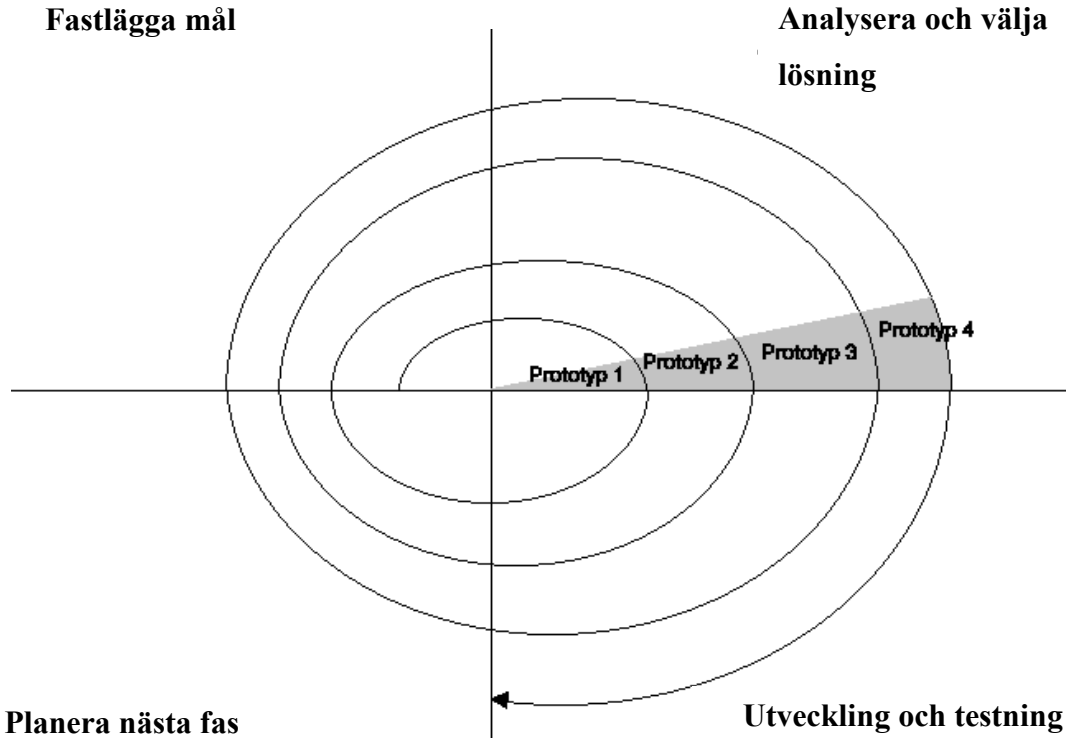
Figur 2.5: Evolutionär utveckling

Problemet med denna modell är att det är svårt att specificera var i utvecklingsprocessen projektet befinner sig. Modellen kan också skapa osammanhängande dokumentation och instabila system eftersom det ständigt sker förändringar.

2.3.3 Spiralmodellen

En modell som är mer anpassad till verkligheten är den så kallade spiralmodellen (se Figur 2.6). Spiralmodellen är en riskdriven process som innebär en upprepning av olika faser [2]. I varje fas finns risker, dessa risker analyseras och bedöms. I varje fas görs också en utvärdering av vilken lösning som innebär minst risk. Med risk menas i det här sammanhanget

sannolikheten att oönskade händelser inträffar. En risk kan vara att kunnig personal faller bort under projektet, en annan att utvecklingen ställer höga krav på hårdvaran, som i sin tur inte klarar av de ställda kraven. Riskerna utvärderas och lösningar på problemen presenteras. Risker/problem och lösningar rankas sedan från ett till tio, där tio betyder lägst risk.



Figur 2.6: Spiralmodellen

Spiralmodellen består av fyra faser enligt [16] (se Figur 2.6):

1. Fastställa mål och risker
2. Analysera och välja lösning med minst risk
3. Utveckla och verifiera lösningen
4. Planera inför nästa loop i spiralen.

Faserna utförs alltså inte sekventiellt med denna modell. Trots detta kan varje ny loop i spiralen representera ett nytt område i processen.
Den största skillnaden mellan spiralmodellen och andra modeller är att kunskap om riskerna i utvecklingen används för att ta olika beslut.

2.4 Jackson System Development

I denna och de två efterföljande avsnitten kommer jag att beskriva tre metoder inom Software Engineering. Den första, Jackson System Development är en lite äldre metod, medan de två senare är stora inom SE idag.

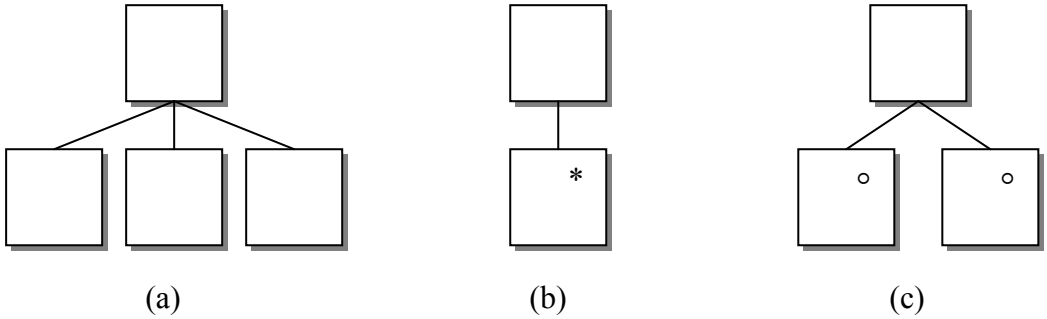
Jackson System Development [8] (JSD) är den äldsta av de tre metoder/processer jag beskriver, JSD utvecklades 1983 av Michael Jackson. Det är en mycket erkänd och populär metod som bygger på Jackson structured programming (JSP). JSD omfattar hela livscykeln från krav till implementation till skillnad från JSP som endast modellerar implementationsfasen. Systemkonstruktion med JSD [8], går i en obruten kedja från att identifiera och modellera användarens verklighet till att få ett körbart system som motsvarar den verkligheten. Eftersom objekten i modellen motsvarar entiteter i verkligheten blir det lättare att förändra och bygga ut systemet.

2.4.1 Strukturen i JSD

JSD delar in utvecklingen av mjukvara i tre stadier:

1. Modelleringsstadiet (Analys av verkligheten)
2. Nätverksstadiet (Systemkonstruktion)
3. Implementationsstadiet (Realisering av modellen)

Steg ett hanterar analysen av den verkliga världen, och att dela upp den i entiteter. Det är mycket viktigt att utvecklaren och kunden kan samarbeta och att utvecklaren kan ställa så detaljerade frågor som möjligt. Entiteterna är objekt i verkligheten och de händelser som påverkar dessa objekt. Entiteterna byggs upp av de händelser som de utsätts för. Varje händelse som kan påverka ett objekt ordnas efter tid, detta ger en överblick av den process som objektet ingår i. Händelserna ordnas sedan efter en rollstruktur som bygger på notation från JSP. Rollstrukturen är ett diagram *(Process Structure Diagram)* som bygger på sekvens, iteration och selektion (se Figur 2.7). Resultatet av modelleringsstadiet är ett antal tabeller och diagram. Dessa diagram talar å ena sidan om vad som är viktigt för systemet ur användarens synpunkt och å andra sidan visar vilken information som är viktig ur teknisk synpunkt och hur den ska lagras.



Figur 2.7: PSD diagram (a) Sekvens, (b) Iteration, (c) Selektion

Steg två sätter samman de processer som associeras med en entitet i ett *systemspecifikationsdiagram* (SSD). SSD är ett diagram över hur olika processer kommunicerar. Steget handlar om att koordinera och synkronisera händelserna i modellen så att de överensstämmer med verkligheten. Detta steg skapar en mer precis bild av hur utdata ska se ut och hur det ska åstadkommas.

Steg tre är implementationsfasen där nätverket i steg två omvandlas till implementation. Nätverket av processer översätts till en sekvens som kan exekveras av en vanlig dator. I denna metod skrivs inga specifika testfall innan implementationen börjar. Testningen utförs istället utifrån de händelser och indata som systemutvecklaren och användaren har kommit fram till.

2.5 Extreme Programming

Extreme Programming (XP) [1] är en disciplinerad metod för att utveckla mjukvara, metoden har sitt ursprung i det tidiga 90-talet. XP bygger på principer som lagarbete, anpassning efter förändrade krav och att kunden ska få sin produkt levererad och klar i tid. [1] säger att utvecklingen av mjukvara kan liknas med att köra bil, om föraren planerar resan och väljer väg kommer han fortare till målet än att bara köra ”på känsla”. Motivationen bakom XP är ungefär samma som de bakom UP, det vill säga ständigt förändrade krav och försenade produkter. Ytterligare likheter är att XP också driver utvecklingen med avseende på risk och att utvecklingen utförs i iterationer. Skillnaden mellan dessa metoder är att XP vänder sig till mindre utvecklingsteam, rekommenderad storlek är mellan 2 och 12 personer. Detta är en av anledningarna till att XP benämns som en ”lättviktsmetod”.

2.5.1 Strukturen i XP

XP bygger på en mängd regler och praxis för hur utveckling av mjukvara ska planeras och utföras. Dessa regler och praxis har ingen signifikans om de står för sig själva, men tillsammans skapar de en kraftfull men ändå lättarbetad metod. XP fokuserar på enkla lösningar; det är bättre att skapa enkla lättförstådda system som enkelt kan byggs ut, än att anlita massor av arbetskraft till ”smarta lösningar” som kräver mer förståelse och mindre hårdvara. I dag läggs stora summor på att skapa, underhålla och uppdatera datasystem samtidigt som det ska sparas in på inköp av snabbare hårdvara. Om mjukvaruutvecklare satsade på enklare lösningar och mindre tekniska knep skulle mer pengar sparas än vad som idag sparas på hårdvaran eftersom systemen lättare skulle kunna byggas ut. XP fokuserar även på testning, testerna ska vara automatiserade och ska skapas både innan, under och efter implementationen. Detta gör att buggar sällan släpps igenom till kunden. Om en bugg hittas skrivs ett nytt testfall och buggen tas omhand.

XP:s struktur bygger på användarstories, små leveranser, iterationer, parprogrammering och planeringen av alla dessa. Så kallade användarstories skapas av utvecklaren tillsammans med kunden. Dessa stories är ganska lika användningsfall men ligger till grund för hur leveransplaneringen ska ske till skillnad från användningsfall som ligger till grund för kravspecifikationen. Utvecklaren gör sedan en planering utifrån de stories som tagits fram, den planering som då görs är hur lång tid varje story beräknas ta att implementera. Implementationen av en story ska ligga mellan 1 till 3 veckor. Om den beräknas ta kortare tid

är storien för detaljerad, om den beräknas ta längre tid så behöver den brytas ned. Översättningen från stories till implementation sker i individuella tasks och översättning sker också till acceptanstester som visar om en story är färdig eller ej. Utefter planeringen av stories planeras leveranserna, en leverans görs varje gång en ny funktionalitet har lagts till. För varje leverans finns ett leveransplaneringsmöte där teamet bestämmer vilka stories som ska implementeras i nästkommande iteration. Parprogrammering bygger på att två utvecklare tillsammans skapar koden. Under programmeringens gång byter utvecklarna arbetsuppgift genom att viss del av tiden vara den som programmerar och viss del var den som kontrollerar koden som skrivs. Fördelningen mellan arbetsuppgifterna ska vara jämn och byte av uppgift ska ske ofta.

Viktiga delar i XP är:

- Planering – Görs med hjälp av användarstories, leveransplanering och iterationsplanering
- Design – Håll designen enkel
- Kodning – Koda i kontakt med kunden så att nya krav hittas snabbt, parprogrammera, dela koden, och gå hem i tid
- Testning – Enhetstesta, skapa nya testfall vid fel och acceptanstesta varje story

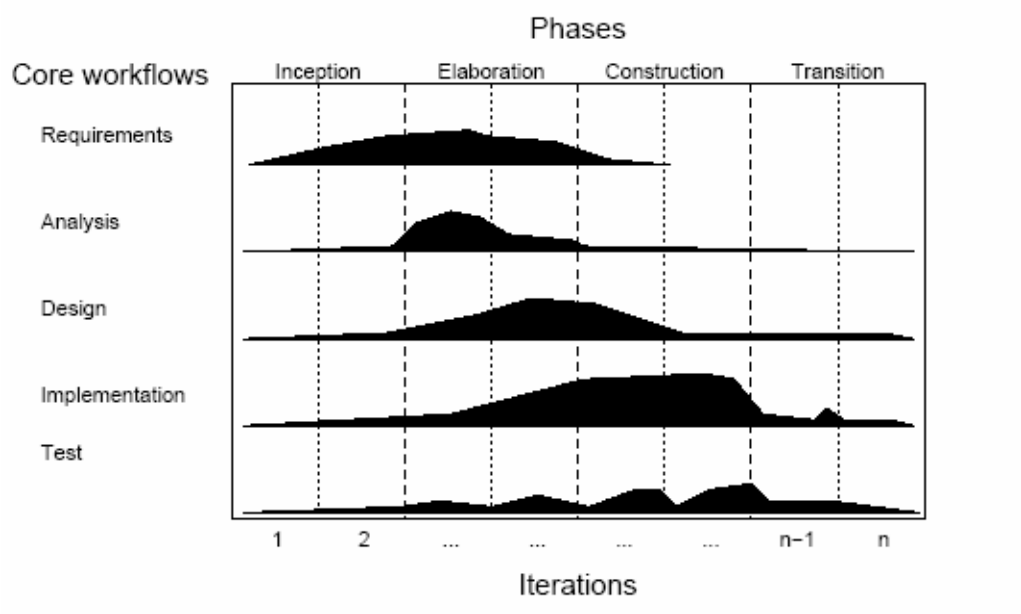
2.6 Unified Process

I detta avsnitt kommer jag att beskriva utvecklingsprocessen Unified Process. Denna beskrivning är mer utförlig än de tidigare två eftersom jag kommer att använda mig av denna process under mitt projekt. Alla metoder inom SE använder många viktiga ord för att beskriva grunderna i metoden. Jag har använt mig av samma översättning som [11] eftersom jag tycker att de är mycket bra.

The Unified Process är en utvecklingsprocess som samlar de aktiviteter som krävs för att föra en användares krav till ett fungerande system. Unified Process är också en generell metod som kan anpassas efter just det ändamål som önskas. Processen skapar ett antal modeller, där användningsfallsmodellen ligger till grund för var och en av de efterföljande modellerna. UP är en omfattande process så jag kommer i följande avsnitt endast att ta upp de centrala momenten.
UP bygger på tre hörnstenar: användningsfallsdriven process, arkitekturcentrerad utveckling och iterativ/inkrementell utveckling. Livscykeln för systemet som ska utvecklas innehåller flera cykler eller iterationer som alla slutar med någon form av produktöverlämning. UP innehåller alla de arbetsmoment som jag beskrivit i avsnitt 2.2: kravinsamling och kravanalys, design, implementation och testning. Dessa utförs i olika faser under utvecklingen och varje fas utförs under ett antal iterationer.

2.6.1 Strukturen i UP

Faserna som ingår i UP är förberedelse (*inception)*, etablering, (*elaboration)*, konstruktion (*construction)*, och överlämning (*transition)*. Hur arbetet med de olika arbetsmomenten kan fördela sig illustreras i Figur 2.8.



Figur 2.8: Faser i UP

Varje fas avslutas i något som kallas en milstolpe; vilken kan användas som underlag för att kontrollera var i utvecklingen projektet befinner sig. Milstolpen innefattar också den produkt som ska kunna levereras efter att fasen är avklarad. Produkten kan vara en ny komponent till systemet eller ett kravdokument som analyserats. Faserna kan beskrivas på följande sätt:

- Förberedelse – kan jämföras med en *feasibility study* beskrivet i avsnitt 2.2. I denna fas ska idéer runt produkten samlas in, vilka visioner som finns och vilka som har intresse i den. En plan för utvecklingen börjar formas här.
- Etablering – Här börjar användningsfallen att ta form, arkitekturen byggs upp, utvecklingsplanen förfinas och kostnaden för produkten kan beräknas mer precist.
- Konstruktion – Implementation och testning sker iterativt så att arkitekturen byggs på och systemet tar form.
- Överlämning – Denna fas kan ses som en form av betatestning av systemet. Utvald personal från kundföretaget testar systemet och rapporterar de defekter eller brister som hittas.

Varje fas görs ett antal gånger för att förfina och analysera vad som gjorts tidigare. Varje iteration skapar ett nytt inkrement på den produkt som ska skapas.

2.6.2 Användningsfallsdriven process

UP är användningsfallsdriven, det vill säga utvecklingen drivs av kraven från användaren. Ett mjukvarusystem skapas för att en användare har ett behov av det och därför är användaren

nyckelpersonen i hela utvecklingen [9]. För att beskriva de funktionella kraven som användaren har används användningsfallsdiagram (se avsnitt 2.2.1). Samtliga användningsfall och funktionella krav ska illustreras med användningsfallsdiagram, summan av dessa diagram bildar användningsfallsmodellen. Användningsfallsmodellen är en av många modeller som skapas under utvecklingsprocessen med UP. Användningsfallen följer hela utvecklingsprocessen, de driver designen, implementationen och testningen. Utvecklingsprocessen följer ett flöde av steg som grundar sig på användningsfallen.

2.6.3 Arkitekturcentrerad process

Arkitektur är lika viktigt inom SE som det är inom andra ingenjörskonster och produkten som ska utvecklas måste modelleras ur flera olika synvinklar. Arkitekturen ger en övergripande bild av hela systemet trots att det inte är färdigkonstruerat. Arkitekturen hjälper därför utvecklaren att se den struktur som måste finnas i systemet, samtidigt som de viktigaste komponenterna kan utses. Den är kopplad till användningsfallen genom att den ger form åt dem, inte bara en funktion. Arkitekturen innebär inte bara realisering av användningsfallen utan även en sammanställning av flera olika faktorer såsom vilken miljö och plattform systemet ska finnas på. Arkitekturen ger inte bara en övergripande bild av hur systemet ska se ut utan också hur olika delar ska samverka. Arkitekturen i UP innehåller ofta olika designmönster eftersom processen ofta används för objektorienterade lösningar.

Arkitekturen utvecklas ofta samtidigt som användningsfallen och detta ställer till ett ”hönan-eller-ägget problem”. Om användningsfallet inte är specificerat hur ska då arkitekturen för det se ut? Problemet löses oftast genom flera iterationer av den pågående fasen, de användningsfall som är färdigspecificerade kan också designas. Arkitekten i projektet startar ofta med de bitar som inte är specifika för ett användningsfall, såsom plattform och ickefunktionella krav.

2.6.4 Iterativ och inkrementell process

Utvecklingen av ett mjukvarusystem kan pågå under flera månaders tid, ibland upp till ett eller flera år. UP delar därför in projektet i miniprojekt eller kontrollerade iterationer vilka resulterar i en påbyggnad av produkten. Dessa iterationer ska vara väl valda och planerade så att de är värdefulla för projektet. Detta kan innebära att välja de användningsfall som är mest relevanta vid en viss tidpunkt samt de användningsfall som hanterar de mest riskfyllda momenten. Efter varje iteration släpps en ny version av produkten, vilket från början är ett

skelett av de modeller som ska ingå i slutprodukten. Slutprodukten innefattar inte enbart den källkod som produceras för att få systemet att fungera, utan även krav, användningsfall, och testfall.

2.6.5 Modeller och Vyer i UP

UP skapar som tidigare nämnts ett antal modeller. Dessa modeller avspeglar varje arbetsflöde (*workflow*) som utvecklingen går igenom. Kravinsamling och kravanalys resulterar i användningsfallsmodellen, arkitektur och design resulterar i designmodellen, implementation i implementationsmodellen, testning i testmodellen och överlämning (*deployment*) i överlämningsmodellen. Varje modell innehåller diagram, dokumentation och till viss del källkod. Arkitekten ser på dessa modeller ur arkitekturens synvinkel och identifierar vad som är intressant för just arkitekturen. Intressanta delar kan vara viktiga användningsfall i användningsfallsmodellen samt viktiga klasser i designmodellen. Dessa synvinklar hjälper till att ge flera olika typer av vyer över projektet.

2.6.6 Modellering med UML

UP använder Unified Modeling Language (UML) [14] för att modellera de olika faserna i processen. UML är idag ett av de största verktygen för modellering inom programvarukonstruktion. UML används för att modellera objektorienterade system och är idag en industristandard. Under projektet i denna uppsats kommer ett antal diagram från UML att presenteras eftersom UP är uppbyggt med hjälp av UML.

2.6.7 Sammanfattning

Inom SE finns många olika metoder, modeller och processer, det är i många fall inte lätt att se skillnad på dem. Att vissa av dem är användbara finns det dock belägg för. Nya avancerade metoder för system- och mjukvaruutveckling resulterar i att produkter finns snabbare på marknaden. De ger också utrymme för användaren och kunden att vara med och utveckla produkten. Idag har dessa fördelar gjort att många större företag har anammat dessa Software Engineering principer och anpassat modeller och processer till sina egna.

Det jag beskrivit om SE i detta kapitel har med själva utvecklingen att göra, jag har inte i mitt arbete gått in på underhåll av systemet eftersom detta inte kommer att ingå i resten av uppsatsen. Det praktiska arbete jag ska utföra sträcker sig fram till överlämning och installation av produkt, inte av något underhåll. Att bortse från den aspekten har varit svårt

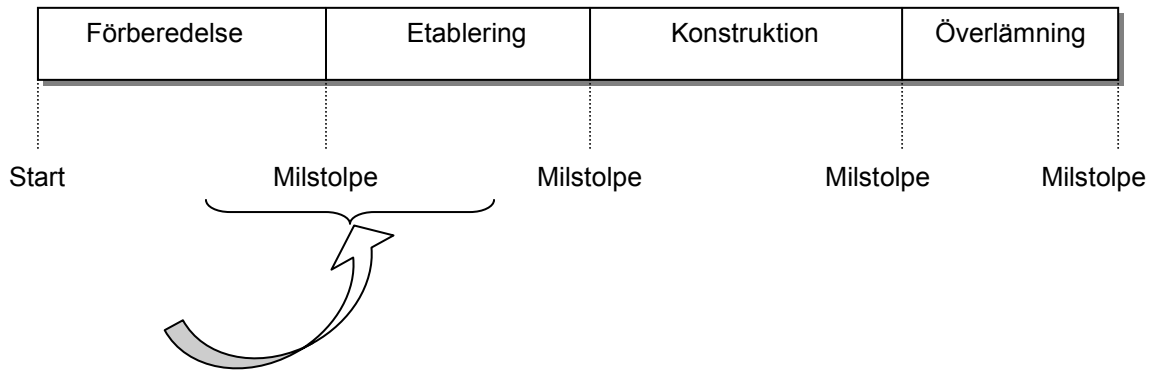
eftersom den stora kostnaden ligger på just underhåll. Inte heller har jag behövt göra någon modell för projektet kostnadsmässigt vilket också är en stor del.

3 Kravinsamling & Användningsfallsmodellen

I detta kapitel presenterar jag de krav som företaget har på det system som ska skapas. Kraven har samlats in genom samtal med användarna. Kapitlet innehåller en beskrivning av funktionella krav, ickefunktionella krav och en sammanställning av kraven i användningsfallsmodellen.

3.1 Kravinsamling

När ett nytt datasystem ska skapas finns det ofta ett stort antal frågetecken, så även i detta fall. Det är till exempel ofta så att det från början inte är klart vilka som i framtiden kommer att använda systemet. En missuppfattning som ofta görs är att om en kund beställer ett system så måste han/hon veta vad systemet ska innehålla och vad det ska kunna utföra. För att räta ut dessa frågetecken görs en kravinsamlig tillsammans med kunden. Kravinsamlingen börjar med något som liknar intervjuer med tänkta användare till systemet. Resultatet av dessa intervjuer omvandlas till användningsfall som specificerar kraven på kundens språk, och därför finns inga implementationsdetaljer med i detta skede. Kraven följer med genom hela utvecklingsprocessen men den största mängden samlas in i etableringsfasen (se Figur 3.1: Nuvarande placering i utvecklingsprocessen).



Figur 3.1: Nuvarande placering i utvecklingsprocessen

3.2 Kravinsamling av funktionella krav

UP är en användningsfallorienterad process och för att skapa användningsfall så måste först de funktionella kraven specificeras. Företaget Ponderator har för närvarande två tänkta användare till det nya systemet och båda har tidigare erfarenhet av hur ett affärssystem kan fungera. Dessa två användare får därför medverka vid intervjuerna.
Kravinsamlingen är till för att svara på de frågor som en utvecklare ställs inför. Jag börjar därför med att ställa frågor för att få en övergripande bild av den funktionalitet som ska finnas i systemet.

Företaget vill ha ett system som kan lagra information om kunder, produkter, offerter, order, och fakturor, samt kopplingarna mellan dessa. Genom denna information kan vi identifiera följande viktiga entiteter i systemet:

- Kund
- Produkt
- Offert
- Order
- Faktura

Användarna vill kunna lägga in information (se attribut i avsnitt 5.7.1) om nya kunder och sedan kunna ändra denna information vid behov. De vill även kunna se en lista på alla kunder. Kunderna är främst företag men även i viss mån privatpersoner. Offerter, order och fakturor ska hanteras på samma sätt i systemet. Produkterna hanteras något annorlunda då företaget bygger sin produktion på gravyr. Priset på en produkt sätts beroende på mängden gravyr. Företaget vill även ha mallar för offerter, order, ordererkännade och fakturor i Microsoft Word och vill kunna överföra information från det nya systemet till denna mall. Utöver dessa krav vill de ha en enkel ren grafisk design där all funktionalitet ska vara lätthanterlig eftersom ingen av användarna är särskilt datakunnig.

De funktionella kraven kan sammanfattas som följer:

- Användaren ska kunna lägga in nya kunder i systemet
- Användaren ska kunna ändra informationen om en kund i systemet
- Användaren ska kunna visa information om en vald kund
- Användaren ska kunna få en lista på alla kunder
- Användaren ska kunna skapa en ny produkt

- Användaren ska kunna ta bort en produkt
- Användarens ska kunna visa alla produkter
- Användaren ska kunna skapa en ny offert
- Användaren ska kunna redigera en offert
- Användaren ska kunna visa information om en vald offert
- Användaren ska kunna visa alla offerter
- Användaren ska kunna skapa en ny order
- Användaren ska kunna redigera en order
- Användaren ska kunna visa information om en vald order
- Användarens ska kunna visa alla order
- Användaren ska kunna skapa en ny faktura
- Användaren ska kunna redigera en faktura
- Användaren ska kunna visa information om en vald faktura
- Användarens ska kunna visa alla fakturor
- Användaren ska kunna öppna en MS Excel-mall och införa en offert, order, ordererkännande och faktura från systemet

3.3 Kravinsamling av ickefunktionella krav

Företaget har inte många icke funktionella krav, mer än att systemet ska fungera i Windows XP miljö. Till att börja med ska systemet endast finnas på en dator. Jag har då i samråd med användarna kommit fram till vilka tekniska hjälpmedel som ska användas. Dessa är:

- Java 1.4.2 för implementation av användargränssnittet.
- MySql Server 4.1.10 för databasen.
- MySql Java Connector JDBC 3.1.7
- MySql ODBC för koppling till MS Office

Java fungerar mycket bra till denna typ av system eftersom det finns färdigt stöd för databasfrågor och uppkoppling. MySql Java Connector behövs för att koppla java-applikationen till databasen.

Företaget hade önskemål om att mallarna som ska användas för order, offerter, och fakturor ska göras i MS Word. Efter att ha diskuterat med dem så kom vi fram till att MS Excel var ett bättre alternativ eftersom det finns stöd för automatisk summering av till exempel fakturans totalsumma. MySql ODBC är kopplingen mellan MS Office och databasen. Frågor kan då ställas direkt i MS Excel med hjälp av MS Query.

3.4 Användningsfallsmodellen

Användningsfallsmodellen används som en överenskommelse mellan utvecklaren och kunden enligt [9]. Den används även som grund i analysmodellen (se avsnitt 4.1). Användningsfallsmodellen innehåller en modell av systemet som i sin tur innehåller aktörer, användningsfall och relationerna där emellan. Redan i användningsfallsmodellen kan för- och eftervillkor för användningsfallen specificeras.

3.4.1 Aktörer

Aktörer är de användare som kommer i kontakt med systemet samt de externa system som används.

Jag identifierar aktörerna i detta fall till:

- Användare – Slutanvändare av systemet
- System – System som presenterar informationen som finns i databasen
- Microsoft Excel – Innehåller en mall för offert, order, ordererkännande och faktura

Eftersom detta är ett litet företag med få delsystem så behöver inte aktörerna någon ytterligare förklaring.

3.4.2 Användningsfall

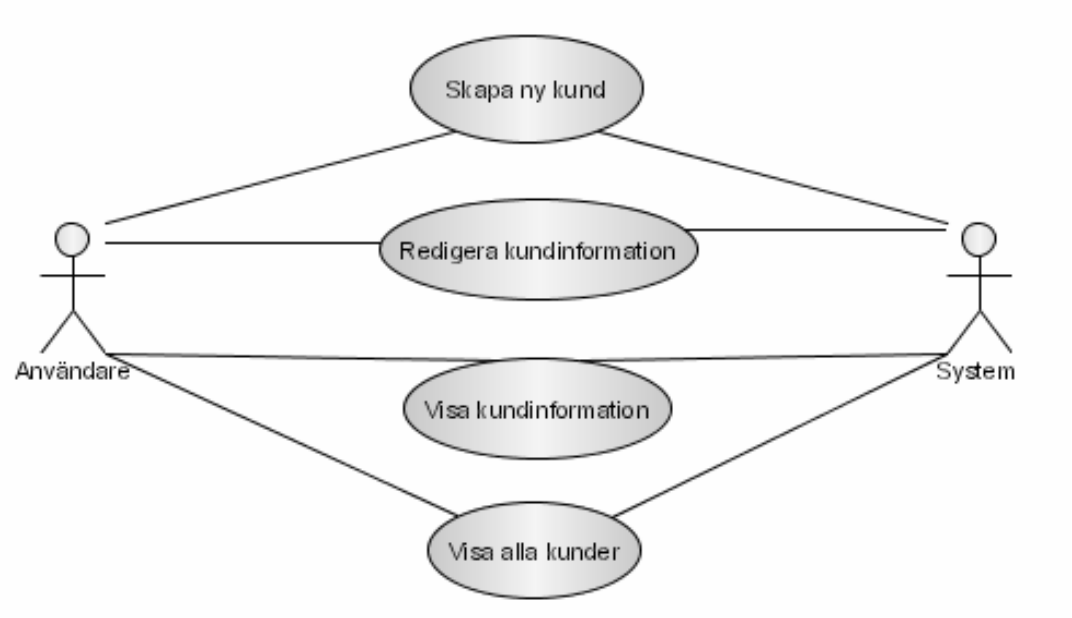
Varje interaktion som en aktör utför i samarbete med systemet är ett användningsfall. Användningsfallen utgår från användaren som utför en handling som påverkar systemet på något sätt. Jag har genom insamlingen av de funktionella kraven kommit fram till ett antal användningsfall. Användningsfallen specificeras i Tabell 3.1, tabellen innehåller användningsfallets namn samt vilka aktörer som berörs. Namnet på ett användningsfall ska vara tydligt och beskrivande.

Användningsfall	Aktör/er
Lägg till ny kund	Användare, System
Ändra kundinformation	Användare, System
Visa information om kund	Användare, System
Visa en lista på alla kunder	Användare, System
Skapa en ny produkt	Användare, System
Ta bort en produkt	Användare, System
Visa alla produkter	Användare, System
Skapa en ny offert	Användare, System
Redigera en offert	Användare, System
Visa information om offert	Användare, System
Visa alla offerter	Användare, System
Skapa en ny order	Användare, System
Redigera en order	Användare, System
Visa information om order	Användare, System
Visa alla order	Användare, System
Skapa en ny faktura	Användare, System
Redigera en faktura.	Användare, System
Visa information om faktura	Användare, System
Visa alla fakturor.	Användare, System
Öppna en MS Excel mall och införa en offert.	Användare, System, MS Excel
Öppna en MS Excel mall och införa en order.	Användare, System, MS Excel
Öppna en MS Excel mall och införa ett ordererkännande.	Användare, System, MS Excel
Öppna en MS Excel mall och införa en faktura.	Användare, System, MS Excel

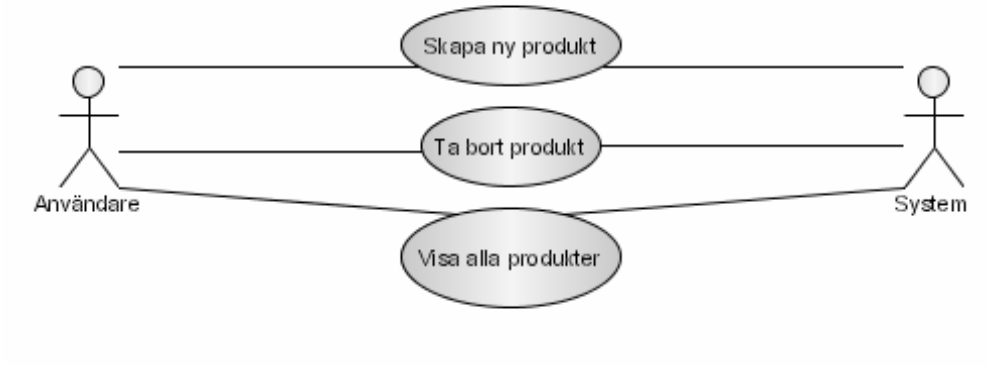
Tabell 3.1: Användningsfall och aktörer

3.4.3 Användningsfallsdiagram

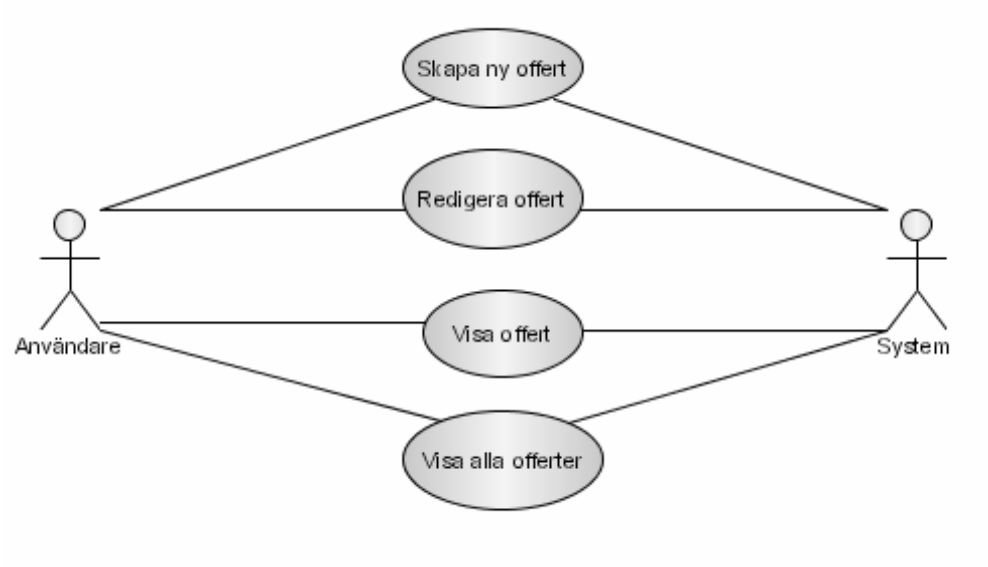
Användningsfallsdiagram illustrerar på ett lättförståeligt sätt de funktionella kraven. Användaren eller kunden kan förstå diagrammen även om ingen tidigare kunskap om UML finns. Nedan följer de användningsfallsdiagram som skapats utefter de krav som samlats in (Se Figur 3.2 - Figur 3.7).



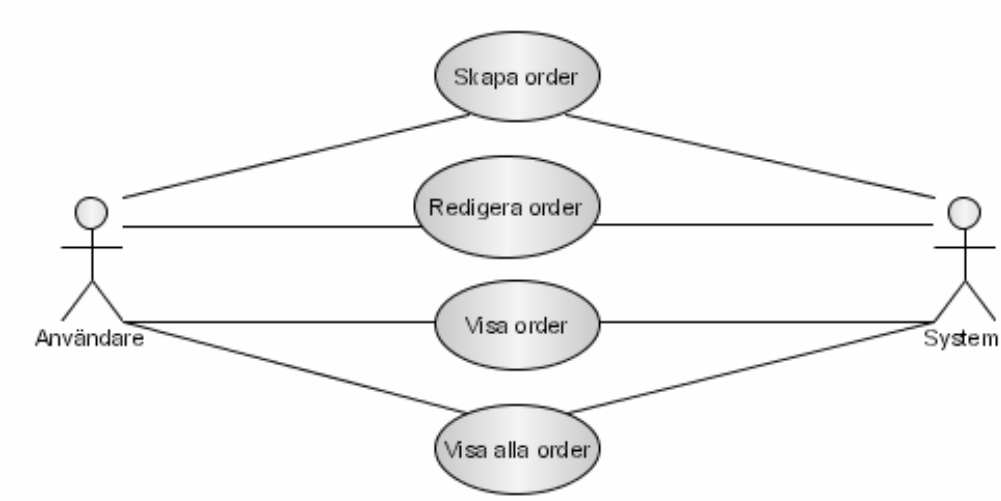
Figur 3.2: Användningsfallsdiagram Kund



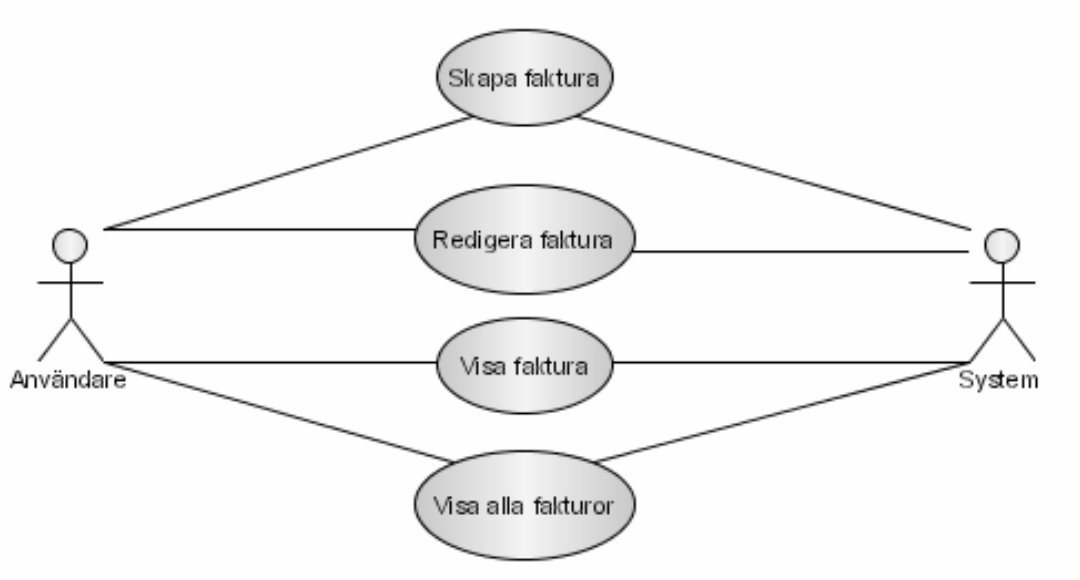
Figur 3.3: Användningsfallsdiagram Produkt



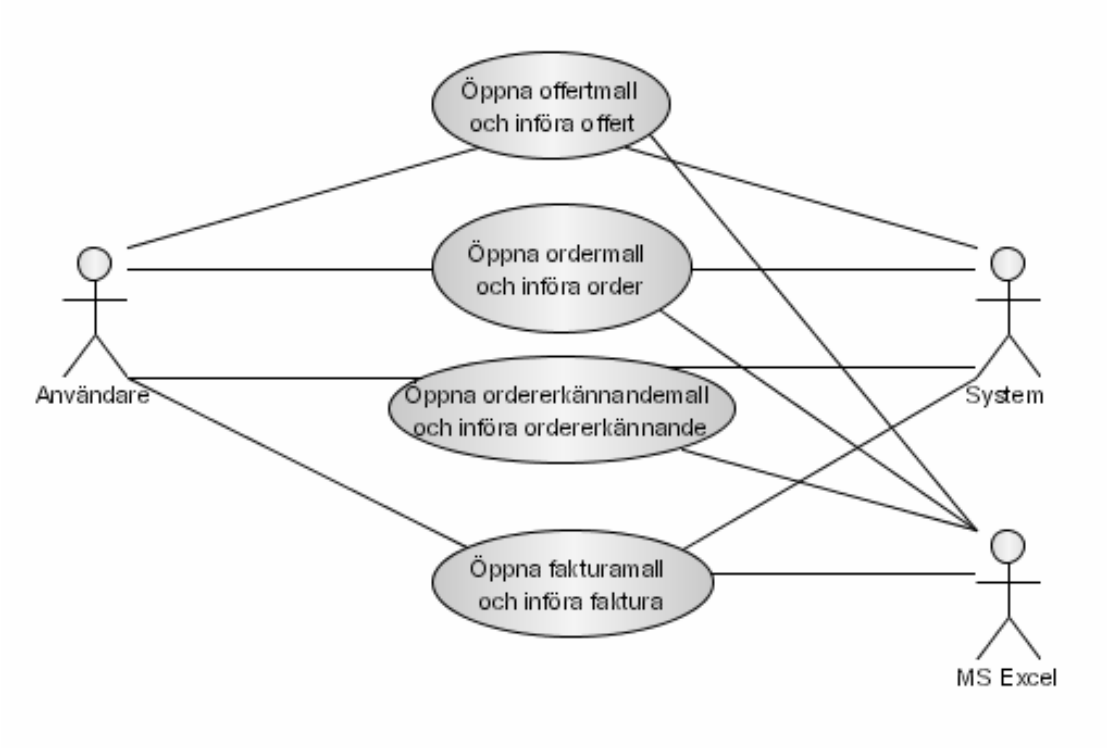
Figur 3.4: Användningsfallsdiagram Offert



Figur 3.5: Användningsfallsdiagram Order



Figur 3.6: Användningsfallsdiagram Faktura



Figur 3.7: Användningsfallsdiagram MS Excel Mall

3.4.4 Specifikation av användningsfall

Jag kommer här att ge en mer specificerad bild av användningsfallen. Specifikationen görs i många fall med hjälp av tillståndsdiagram. Jag använder mig dock inte av dessa eftersom de

ofta krånglar till förståelsen för kunden. De användningsfall vi har identifierat är inte så komplexa i sig så det räcker att beskriva dem i ren text.

Användningsfall:

Lägg till ny kund

1. Användaren öppnar ett nytt kundformulär
2. Användaren skriver in kundinformation
3. Användaren sparar kundinformationen

Redigera kundinformation

1. Användaren matar in kundnamnet
2. Kundinformationen visas i ett editerbart formulär
3. Användaren ändrar kundinformationen
4. Användaren sparar kundinformationen

Visa kundinformation

1. Användaren matar in kundnamnet
2. Kundinformationen visas i ett editerbart formulär

Visa alla kunder

1. Användaren väljer visa alla kunder
2. Användaren får en lista på alla kunder

Lägg till ny produkt

1. Användaren öppnar en ny eller befintlig offert, order, eller faktura
2. Användaren väljer lägg till produkt
3. Användaren skriver in produktinformation
4. Användaren sparar produktinformationen

Ta bort produkt

1. Användaren öppnar en ny eller befintlig offert, order, eller faktura
2. Användaren väljer en produkt
3. Användaren tar bort produkten

Visa alla produkter

1. Användaren väljer visa alla produkter
2. Användaren får en lista på alla produkter

Skapa ny offert

1. Användaren öppnar ett nytt offertformulär
2. Användaren skriver in offerten
3. Användaren sparar offerten

Redigera offert

1. Användaren matar in kundnamnet
2. Användaren väljer offert ur listan som visas
3. Användaren ändrar offerten
4. Användaren sparar offerten

Visa offert

1. Användaren matar in kundnamnet
2. Användaren väljer offert ur listan som visas

Visa alla offerter

1. Användaren väljer visa alla offerter
2. Användaren får en lista på alla offerter

Skapa ny order

1. Användaren öppnar ett nytt orderformulär
2. Användaren skriver in ordern
3. Användaren sparar ordern

Redigera order

1. Användaren matar in kundnamnet
2. Användaren väljer order ur listan som visas
3. Användaren ändrar ordern
4. Användaren sparar ordern

Visa order

1. Användaren matar in kundnamnet
2. Användaren väljer order ur listan som visas

Visa alla order

1. Användaren väljer visa alla order
2. Användaren får en lista på alla order

Skapa ny faktura

1. Användaren öppnar ett nytt fakturaformulär
2. Användaren skriver in fakturan
3. Användaren sparar fakturan

Redigera faktura

1. Användaren matar in kundnamnet
2. Användaren väljer faktura ur listan som visas
3. Användaren ändrar fakturan
4. Användaren sparar fakturan

Visa faktura

1. Användaren matar in kundnamnet
2. Användaren väljer faktura ur listan som visas

Visa alla fakturor

1. Användaren väljer visa alla fakturor
2. Användaren får en lista på alla fakturor

Öppna offertmall

1. Användaren öppnar en offertmall
2. Användaren väljer infoga offert
3. Användaren skriver in offertnummer

Öppna ordermall

1. Användaren öppnar en ordermall
2. Användaren väljer infoga order
3. Användaren skriver in ordernummer

Öppna ordererkännandemall

1. Användaren öppnar en ordererkännandemall
2. Användaren väljer infoga order
3. Användaren skriver in ordernummer

Öppna fakturamall

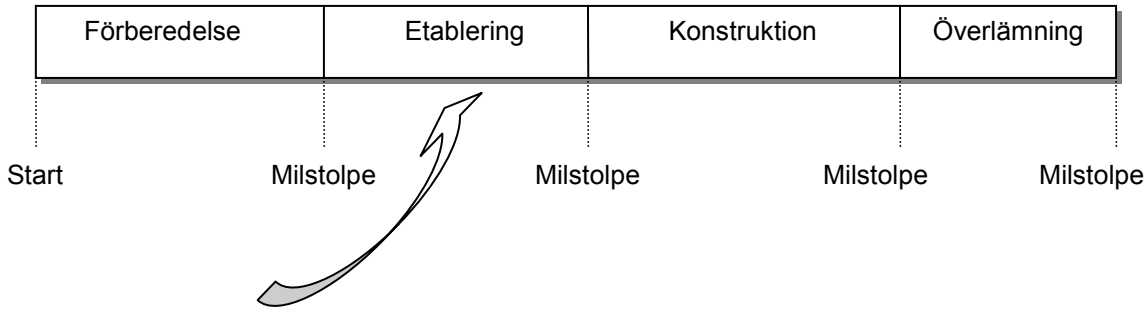
1. Användaren öppnar en fakturamall
2. Användaren väljer infoga faktur
3. Användaren skriver in fakturanummer

4 Kravanalys & Analysmodellen

Detta kapitel innehåller en beskrivning av den kravanalys som skett av de insamlade kraven. Kapitlet ger en översikt av vad som ingår i analysen och beskriver de delar som ingår i analysmodellen. Ett antal samarbetsdiagram visar hur användningsfallen i föregående kapitel förs vidare mot implementation.

4.1 Kravanalys

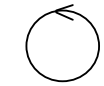
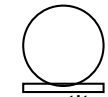
I kravanalysen struktureras och förfinas de krav som samlats in. Kravinsamlingen och användningsfallsmodellen använder användarens språk medan analysen använder utvecklarens språk. Fokus på analysen är i starten på etableringsfasen och analysmodellen är ett av dokumenten som finns med i milstolpen då fasen är avslutad (se Figur 4.1).



Figur 4.1: Nuvarande placering i utvecklingsprocessen

4.2 Analysklasser

I analysmodellen används ett antal analysklasser för att identifiera och stödja de funktionella krav som samlats in. Analysklasserna är av tre typer (se Tabell 4.1):

Beteckning	Klass	Beskrivning
	Boundary class	Klass som visar interaktion mellan systemet och användaren. Hämtar och lämnar ofta olika typer av information.
	Control class	Klass som hanterar transaktioner, samtida händelser och liknande. Delegerar arbete till andra klasser.
	Entity class	Klass som beskriver information som är långlivad. I objektorientering kan det motsvara en faktura.

Tabell 4.1: Analysklasser

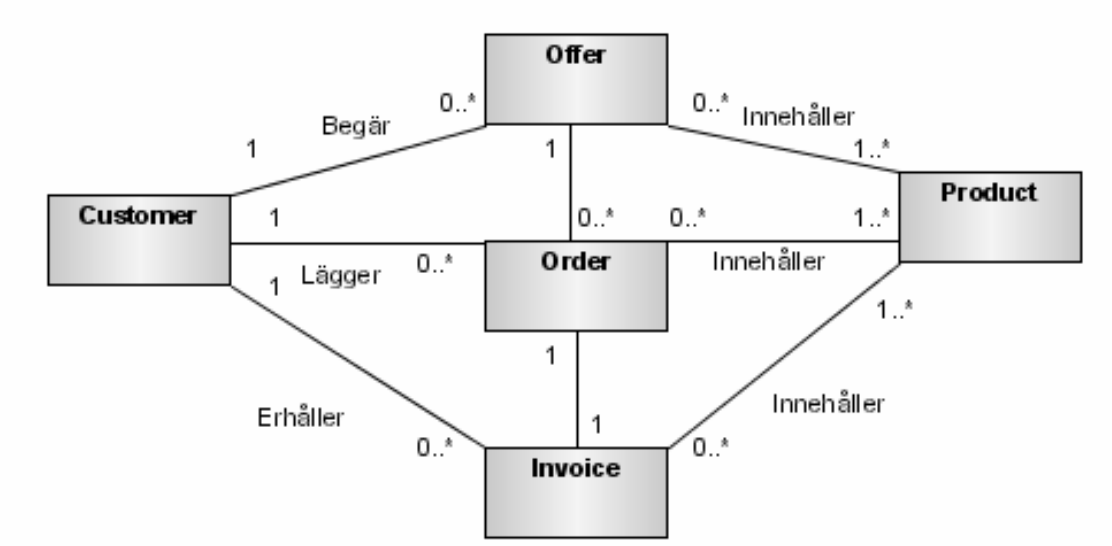
Analysklasserna används för att utföra realiseringen av användningsfallen (*use-case realization*). Analysklasserna är entiteter i projektets problemdomän, alltså klasser som representerar verkliga objekt. Varje användningsfall blir ett samarbetsdiagram bestående av dessa analysklasser.

De analysklasser jag identifierat är:

- GUI: En *boundary class* som hanterar kommunikationen med användaren och controllerklassen
- Controller: Den klass som kontrollerar vilken information som ska hämtas ur och lägga in i databasen, *controller class*. Har också hand om vad som visas i användargränssnittet
- Database: Den databas som sytemet kommer att arbeta mot
- Customer: Entitetsklass som representerar en kund
- Product: Entitetsklass som representerar en produkt
- Offer: Entitetsklass som representerar en offert
- Order: Entitetsklass som representerar en order
- Invoice: Entitetsklass som representerar en faktura

4.3 Konceptuell vy

Systemet kan sammanfattas med en konceptuell vy (se Figur 4.2). Klassdiagrammet visar de entiteter som grundar hela systemet. Kunden ligger till grund för att de andra objekten ska ha en betydelse. Kunden kan begära offerter, lägga ordrar och därefter erhålla fakturor på lagda och levererade ordrar. Offerten kan resultera i en order och en order resulterar i en faktura. Offert, order och faktura innehåller alla objektet produkt.



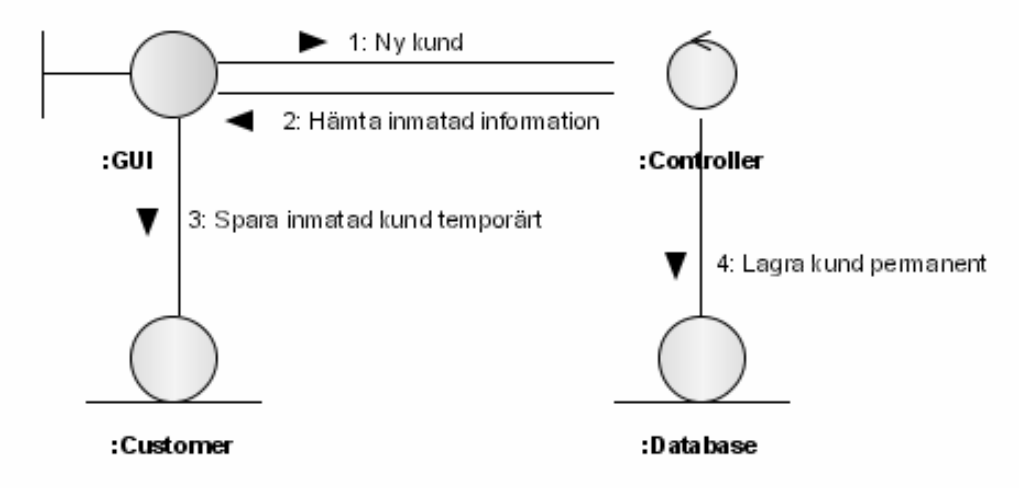
Figur 4.2: Konceptuellt klassdiagram

4.4 Realisering av användningsfall

Jag har enligt UP skapat samarbetsdiagram för alla användningsfallen. Några av dessa presenteras nedan med en beskrivning. Jag har valt att endast presentera en delmängd av de diagram som jag producerat i analysen Eftersom diagrammen som visar skapandet och redigeringen av offert, order och faktura är mycket lika varandra. Även de användningsfall som resulterar i en lista av kunder, produkter, offerter, order, eller fakturor ingår som en del av de tidigare nämnda diagrammen.

4.4.1 Realisering av användningsfall Ny Kund:

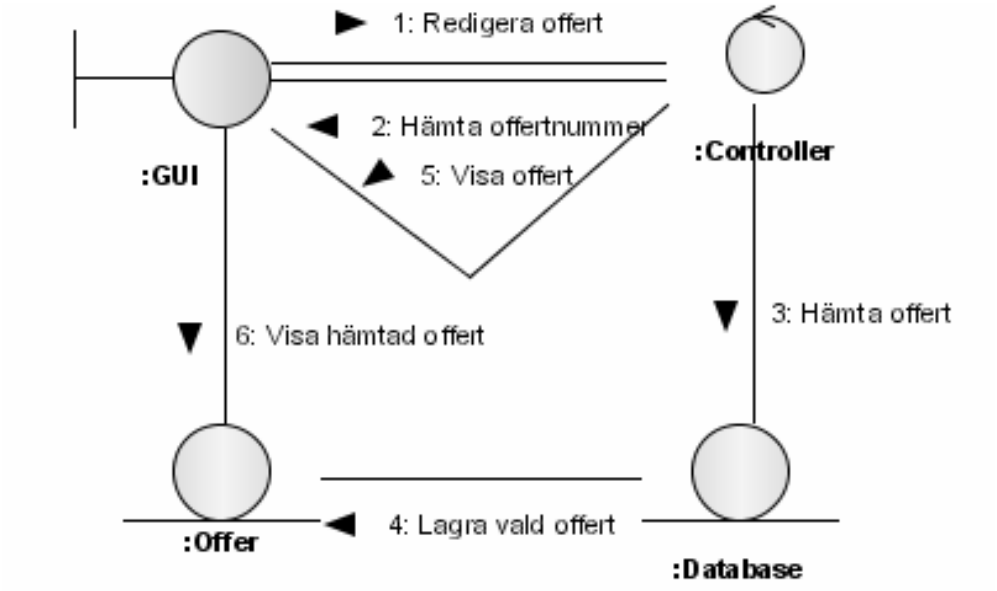
Användaren väljer att skapa en ny kund i användargränssnittet. Användaren skriver in kundinformationen och sparar. Controllerklassen för användargränssnittet hämtar den inskrivna informationen till ett kundobjekt och vidarebefordrar sparandet till controllerklassen för databasen. Denna controllerklass ser till att rätt information hamnar i databasen via gränssnittet till modellen (se Figur 4.3).



Figur 4.3: Samarbetsdiagram Ny kund

4.4.2 Realisering av användningsfall Visa-Redigera Offert:

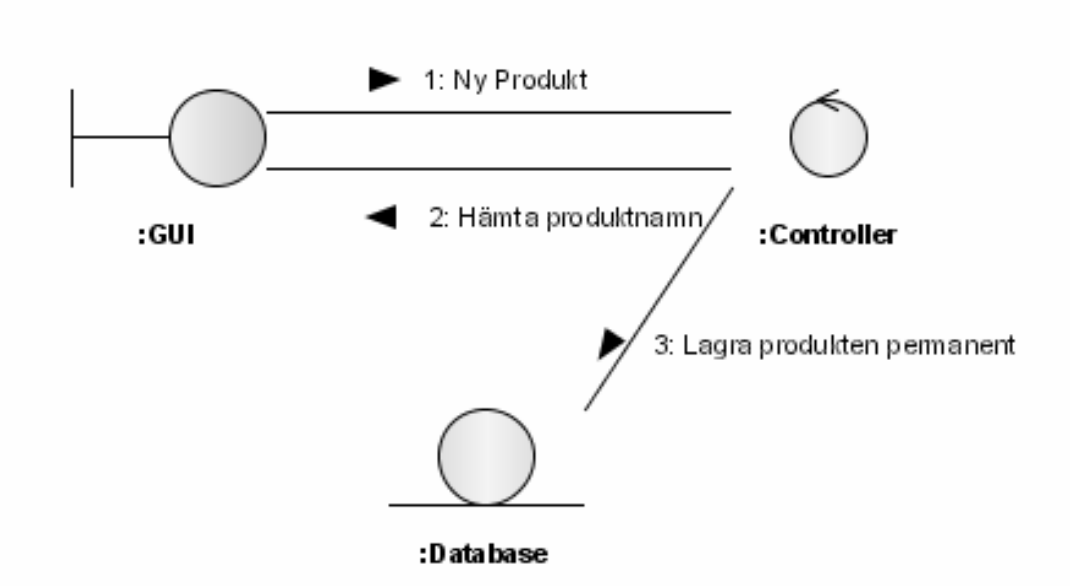
Användaren väljer att visa eller redigera en offert i användargränssnittet, användaren får då skriva in offertnumret. Controllerklassen för användargränssnittet hämtar offerten via databas controller klassen och databasuppkopplingen. Händelsehanteraren för användargränssnittet skapar sedan ett nytt grafiskt objekt för att visa offerten (se Figur 4.4).



Figur 4.4: Samarbetsdiagram Visa-Redigera Offert

4.4.3 Realisering av användningsfall Ny Produkt:

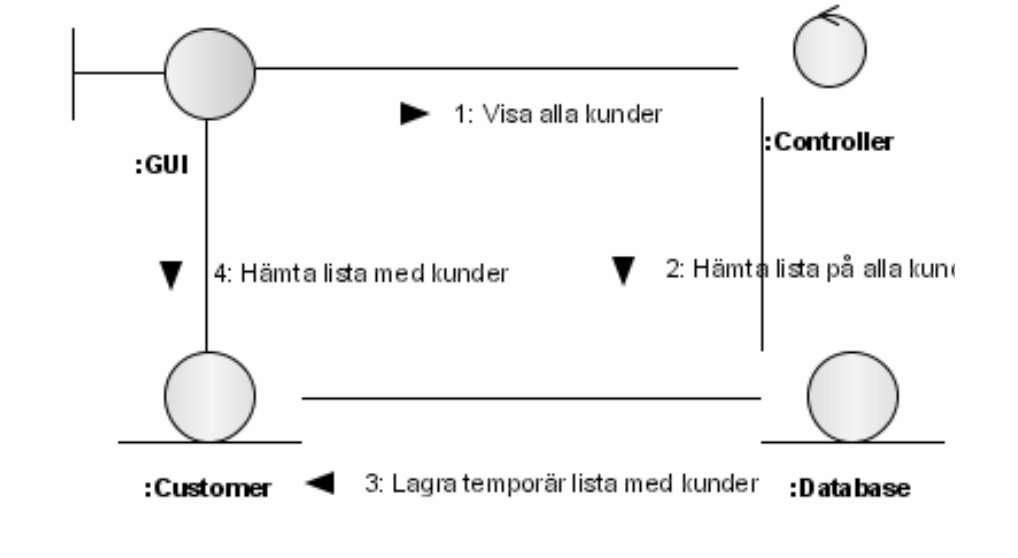
Användaren väljer att skapa en ny produkt. Händelsehanteraren visar då ett objekt för att skriva in produktnamnet. Användaren skriver in produktnamnet och väljer att lägga till produkten. Händelsehanteraren och databashanteraren tar hand om sparande på samma sätt som ”ny kund” (se Figur 4.3 och Figur 4.5).



Figur 4.5: Samarbetsdiagram Ny Produkt

4.4.4 Realisering av användningsfall Visa alla kunder:

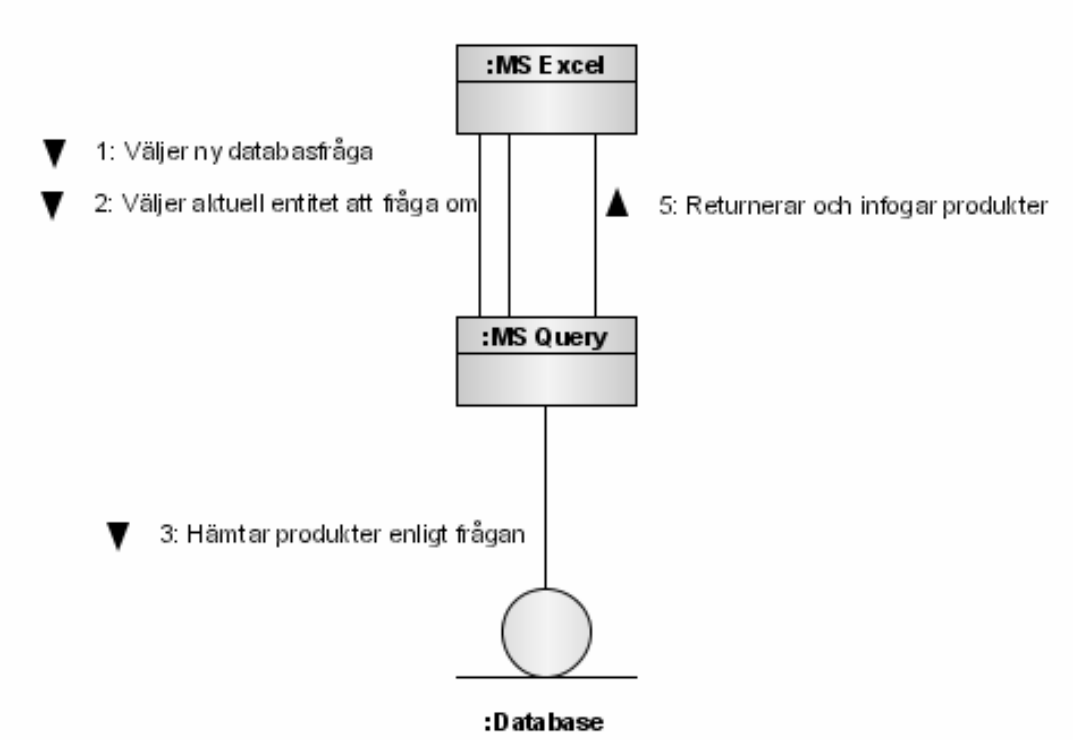
Användaren väljer att visa alla kunder i grässnittets huvudfönster. Controllerklassen för gränssnittet vidarebefordrar önskemålet om att alla kunder ska visas till databas hanteraren och databasuppkopplingen. Controllern för användargränssnittet skapar sedan ett grafiskt objekt för att visa alla kunder (se Figur 4.6).



Figur 4.6: Samarbetsdiagram Visa alla kunder

4.4.5 Realisering av användningsfall Skapa dokument

Användaren öppnar en färdig Excel-mall och skriver in kundinformation samt datum. Användaren tar hjälp av MS Query för att föra in de produkter som finns i fakturan i databasen. Användaren lägger in produktlistan och sparar fakturan se Figur 4.7.



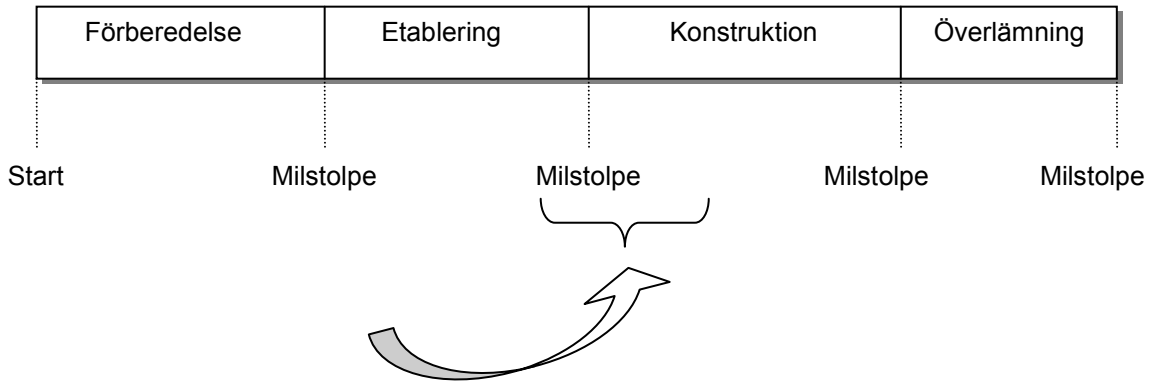
Figur 4.7: Samarbetesdiagram skapa dokument

5 Designmodellen & Databasen

I detta kapitel beskriver jag hur designen för systemet ser ut. Jag beskriver den modell som använts för att skapa systemet (MVC) samt det gränssnitt som identifierats för den modellen. Kapitlet innehåller även designmodellen som skapats enligt UP. Jag har även beskrivit designen av de mallar som ska skapas åt företaget. Sist i kapitlet beskrivs designen av databasen som är kopplad till systemet.

5.1 Design

Designmodellen har som mål att ge en bild av systemets struktur. I detta skede används ett mer formellt språk samtidigt som implementationsdetaljer tas upp. Designen har till stor del att göra med de ickefunktionella krav som kunden har på systemet.



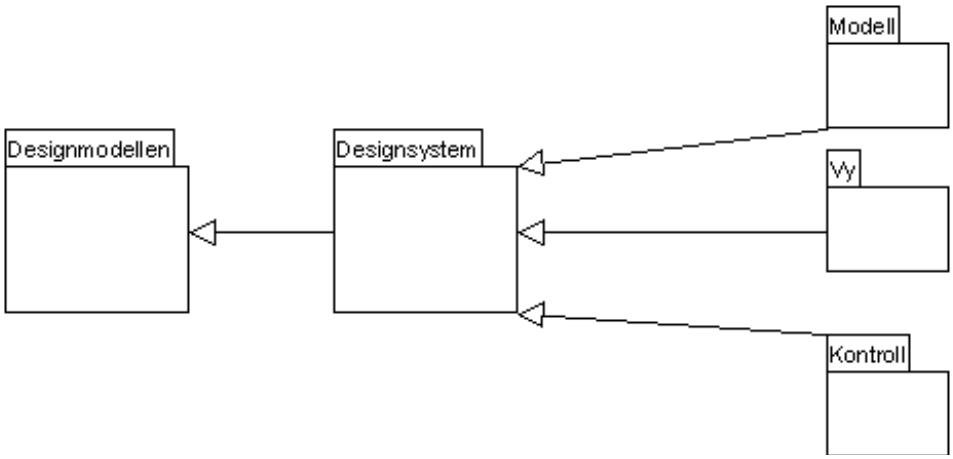
Figur 5.1 Nuvarande placering i utvecklingsprocessen

Designmodellen innehåller precis som analysmodellen ett antal diagram. I designfasen är diagrammen till för att illustrera den övergripande strukturen samt hur systemet delas in i mer hanterbara komponenter. Modellen representeras av ett designsystem som är uppbyggt av subsystem och komponenter. Designmodellen innehåller även sekvensdiagram för att illustrera mer i detalj hur användningsfallen ska realiseras. I denna fas fokuseras även arbetet på återanvändbarhet och utbyggbarhet.

5.2 Designsystemet

Designmodellen och flera andra modeller inom UP representeras genom ett system, i detta fall ett designsystem. Designsystemet är till för att skapa en struktur av de artefakter som finns med i designen. Fördelen med att skapa en struktur är att olika utvecklare och arkitekter kan arbeta med var sin del av systemet samtidigt.

Designsystemet består av subsystem. Dessa subsystem kan rekursivt innehålla fler subsystem eller någon annan typ av komponent. Olika typer av komponenter kan vara designklasser, realisering av användningsfall, gränssnitt eller diagram. Dessa subsystem och komponenter mappas sedan till subsystem och komponenter i implementationsmodellen (se kapitel 8). Designsystemet jag skapat visas i Figur 5.2.



Figur 5.2: Designsystemet

5.2.1 Subsystem

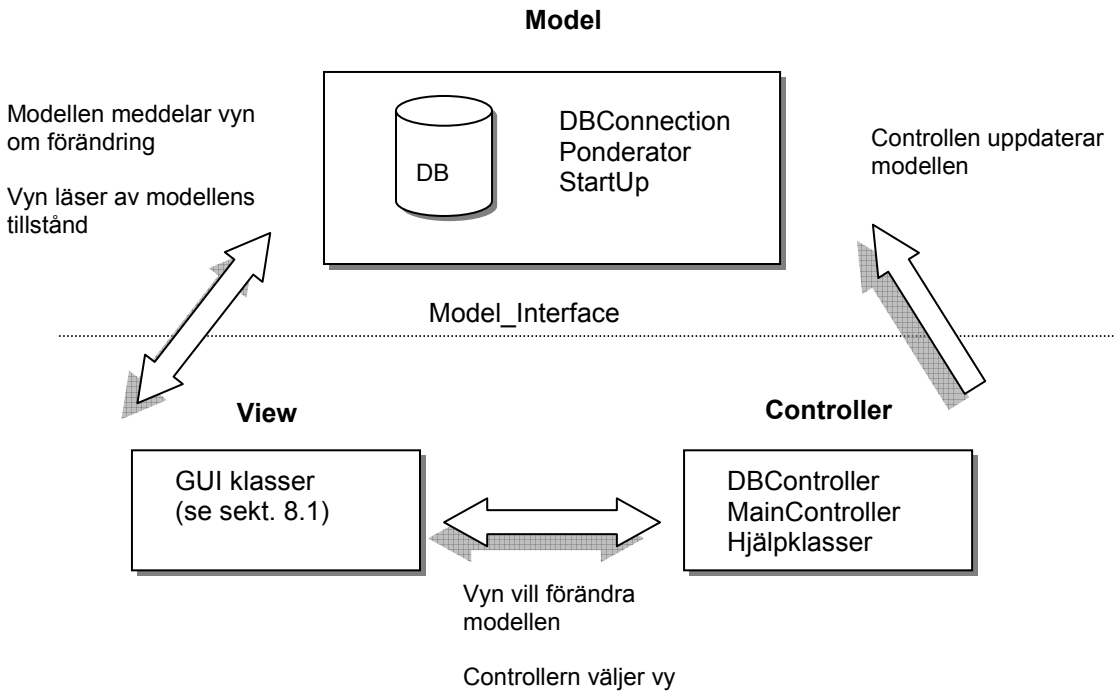
Subsystemen som finns i designsystemet är:
Modell – innehåller databas och databasuppkoppling som egna subsystem
Vy – innehåller kund, offert, order, faktura, produkt som egna subsystem
Kontroll- innehåller kontroll för kopplingen mellan vy och modell

5.3 Övergripande design

Systemet som ska skapas är relativt litet ur designsynpunkt men trots detta finns det krav på att det ska vara lätt att bygga ut. Företaget har just startat och kanske kommer det kräva ett större system i framtiden. Systemet har tre viktiga komponenter: användargränssnittet (se avsnitt 6.3), databasen (se avsnitt 5.7) samt kopplingen mellan dessa (se avsnitt 6.4).

5.3.1 Model-View-Controller modellen

Model-View-Controller modellen bygger på att systemets vy frikopplas från systemets modell [3]. Detta görs för att det ska vara lätt att byta vy och modell samtidigt som systemet lätt ska kunna byggas ut. I detta fall bildar användargränssnittet vyn, databasen modellen och funktionaliteten där emellan kontrollen. I vyn ingår alla grafiska klasser, dessa får endast kommunicera med modellen genom set och get funktioner som finns i gränssnittet. Kontrollen innehåller kontrollklasser för händelsehantering i användargränssnittet samt kontrollklass för databasen. Mellan modellen och kontrollen finns ett gränssnitt definierat. Detta gränssnitt innehåller de metoder som kontrollen och vyn får anropa (se Figur 5.3).



Figur 5.3: Model_View_Controller design

Modellen innehåller databasen och uppkopplingen till denna. Utöver detta lagrar systemets huvudklass Ponderator de aktuella entiteterna som systemet arbetar med för tillfället. Det vill säga, att om användaren vill öppna informationen om en viss kund så skapas ett objekt ”Customer” som lagras i klassen Ponderator. Dessa kommer kontrollen och vyn åt genom gränssnittet. Vyn får använda entitetsklasserna som hjälpklasser men de är endast temporära lagringsstationer mellan anrop.
Vyn har ingen annan uppgift än att ta emot information från användaren och visa information för användaren. All funktionalitet ligger i kontrollklasserna.

5.3.2 Gränssnittet mot modellen

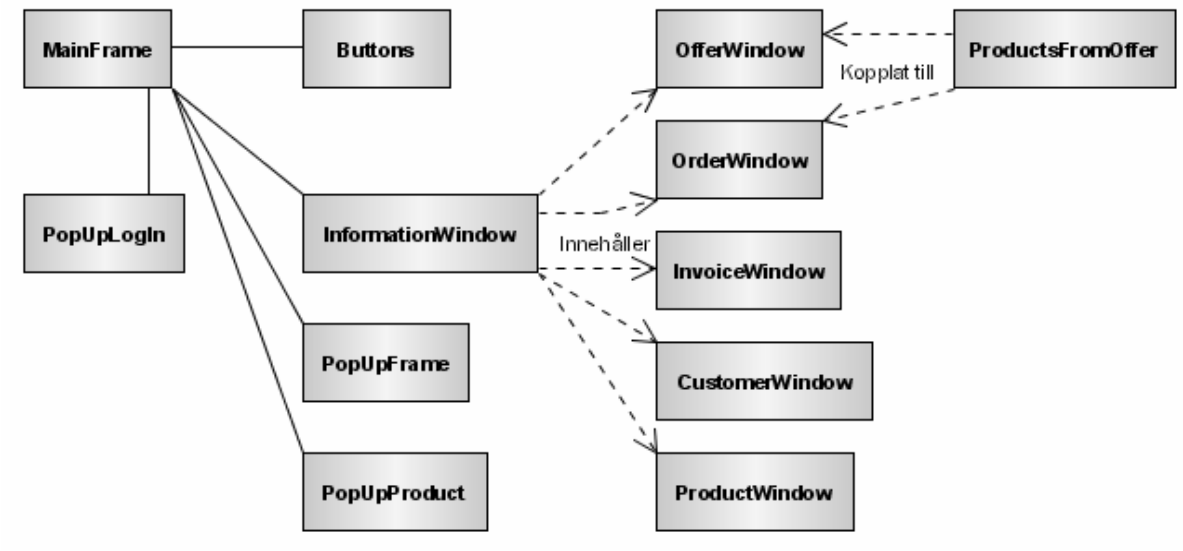
Något som är viktigt i designfasen är att identifiera viktiga grässnitt i systemet. I detta fall kommer ett gränssnitt att användas. Gränssnittet har jag döpt till Model_Interface eftersom det motsvarar metoder i modellen. Gränssnittet innehåller metoder för att hämta och lämna information till klasser i modellen. Dessa metoder gäller både access till databasen och till de aktiva objekten i modellen. Gränssnittet ger vyn och kontrollen access till modellen, men genom ett kontrollerat sätt. Uppkopplingsfunktioner och exekvering av frågor görs internt i modellen.

5.4 Klassdiagram

Efter uppdelningen av systemet i modell, vy och kontroll så har dessa delar individuellt designats. Klassdiagram och en beskrivning av dessa följer i det tre nästkommande avsnitten. Användandet av klasserna kan ses i avsnitt 5.5.

5.4.1 Vy

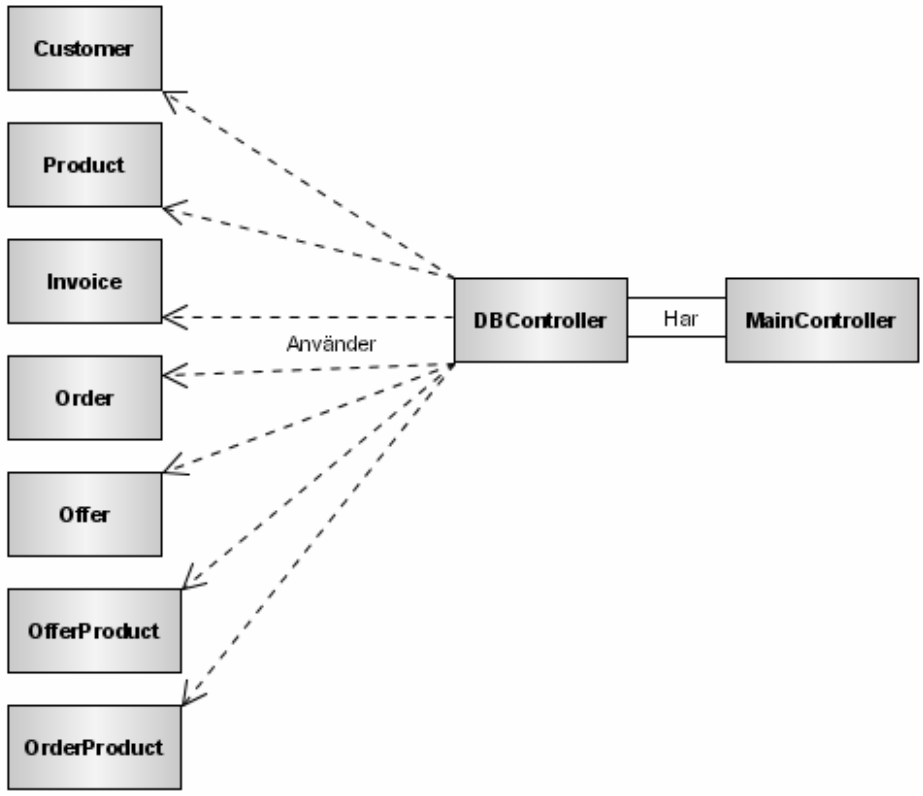
Vyn består av ett huvudfönster MainFrame som visas vid uppstart (Efter inloggning i PopUpLogIn). Huvudfönstret består av ett informationsfönster som innehåller en underliggande panel beroende av vad användaren vill göra. Dessa underliggande paneler består av kund, offert, order, eller faktura fönster. PopupProduct visas då en produkt ska läggas till. PopUpFrame visas då ett objekt ska plockas upp genom ett unikt nummer. ProductsFromOffer visas då en order eller faktura skapas och produkter från en offert eller order ska väljas.



Figur 5.4: Klassdiagram Vy

5.4.2 Kontroll

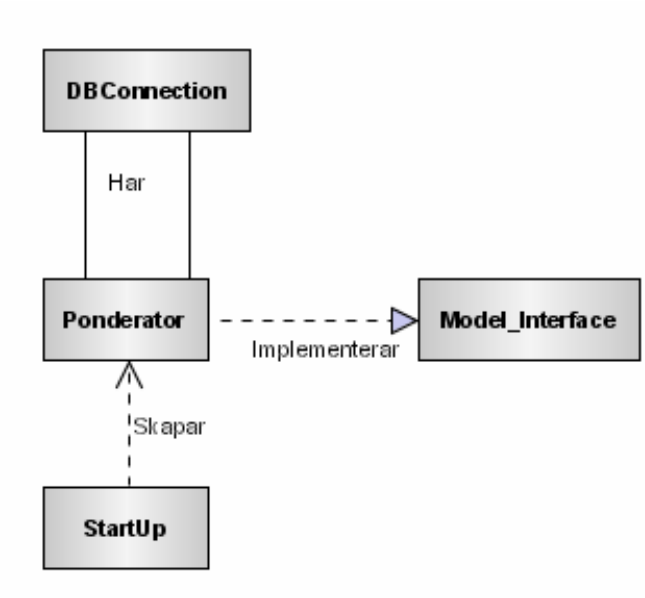
Kontrollen består av MainController som är händelsehanterare för de grafiska komponenterna. DBController är kontrollenhet för databasen. Det är i denna klass som Sql-frågorna skapas. Paketet innehåller även ett antal hjälpklasser (Customer med flera).



Figur 5.5: Klassdiagram Kontroll

5.4.3 Modell

Modellen innehåller förutom databasen även databasuppkopplingen. Här finns även systemets gränssnitt mot modellen samt Ponderator som implementerar detta gränssnitt. StarUp är systemets startklass.



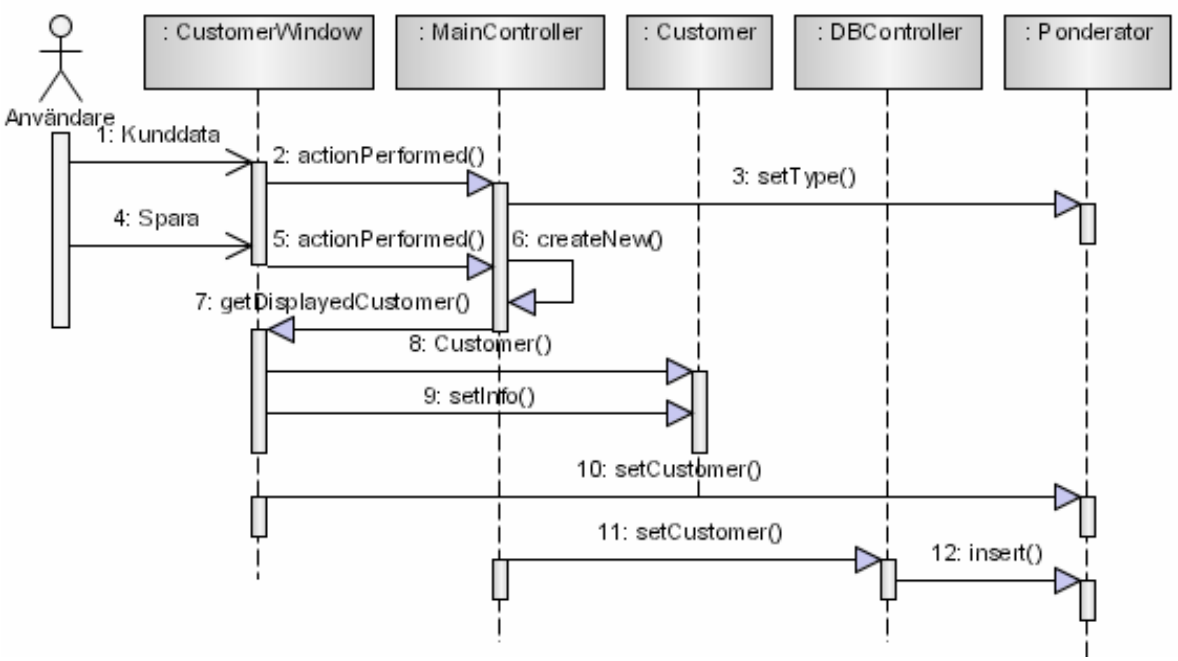
Figur 5.6: Klassdiagram Modell

5.5 Sekvensdiagram

Varje användningsfall får ett sekvensdiagram som illustrerar hur de olika komponenterna i systemet ska uppnå de funktionella kraven. Jag kommer här att referera till fyra sekvensdiagram som följer de fyra samarbetsdiagram som visades i avsnitt 4.4.

5.5.1 Ny kund

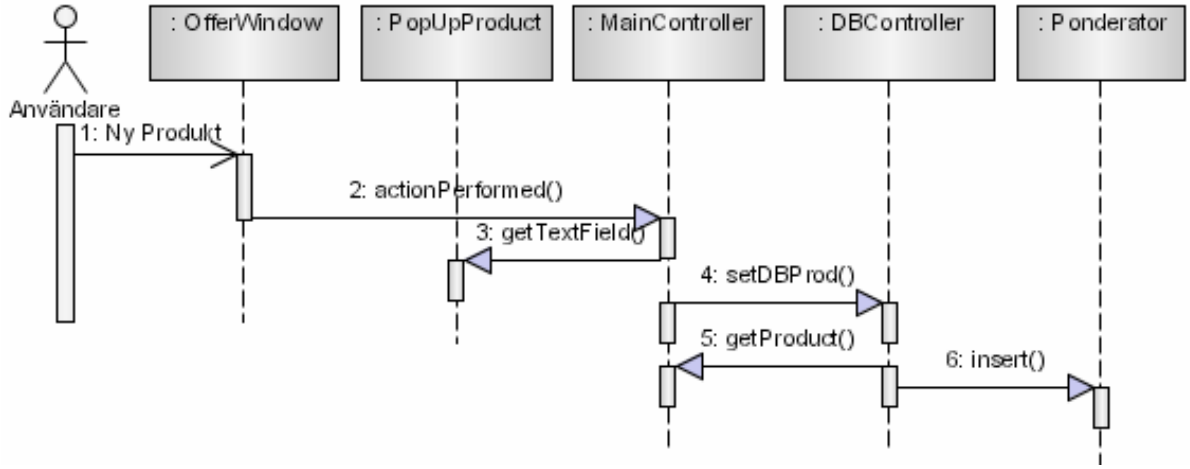
Sekvensdiagrammet visar hur användaren väljer att skapa en ny kund i huvudfönstret (se Figur 5.7). Händelsehanteraren MainController tar hand om öppnandet av ett nytt kundformulär. Användaren skriver sedan in kunddata såsom namn, adress, telefonnummer och så vidare. När användaren trycker på spara, mottar MainController händelsen igen och hämtar informationen som finns i fönstret. En hjälpklass Customer skapas som temporärt lagrar kunddata. Kontrollklassen för databasen tar då vid och sparar informationen som en ny kund i modellen.



Figur 5.7: Sekvensdiagram Ny Kund

5.5.2 Ny Produkt

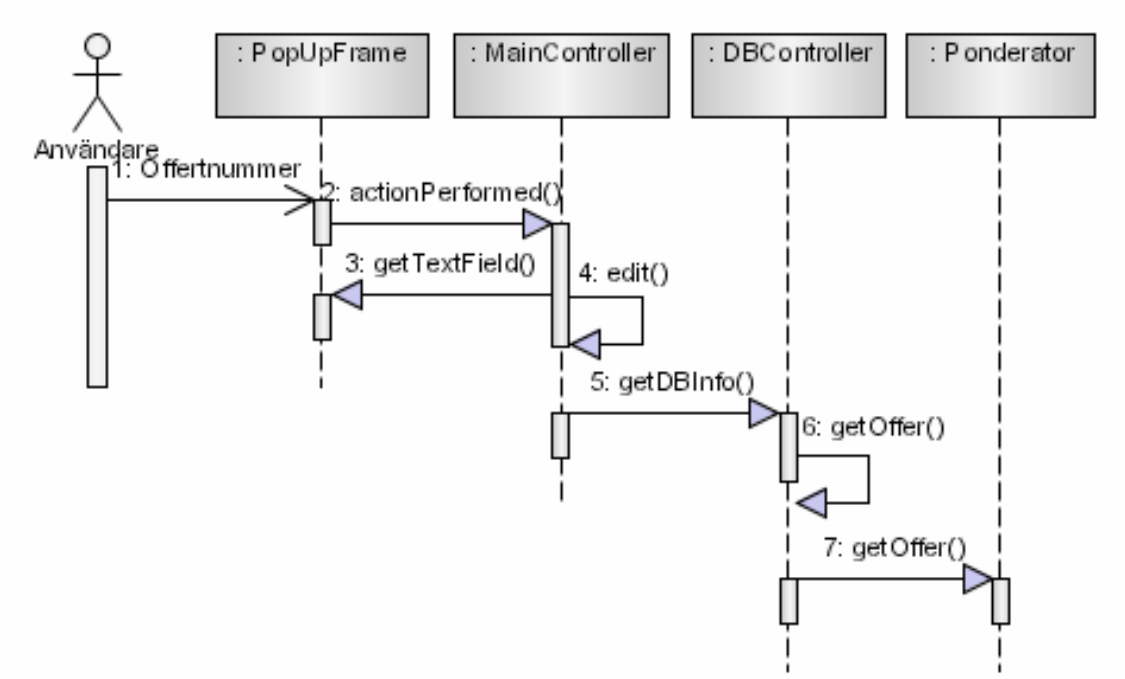
Sekvensdiagrammet Ny Produkt visar hur användaren väljer att skapa en ny produktpost (se Figur 5.8). Användaren öppnar en befintlig offert, order eller faktura och väljer sedan att skapa en ny produkt. Händelsehanteraren reagerar genom att skapa ett popupfönster så att användaren kan skriva in produktnamnet. Sparandet av produkten sker på samma sätt som sparandet av kunden ovan.



Figur 5.8: Sekvensdiagram Ny Produkt

5.5.3 Visa-Redigera Offert

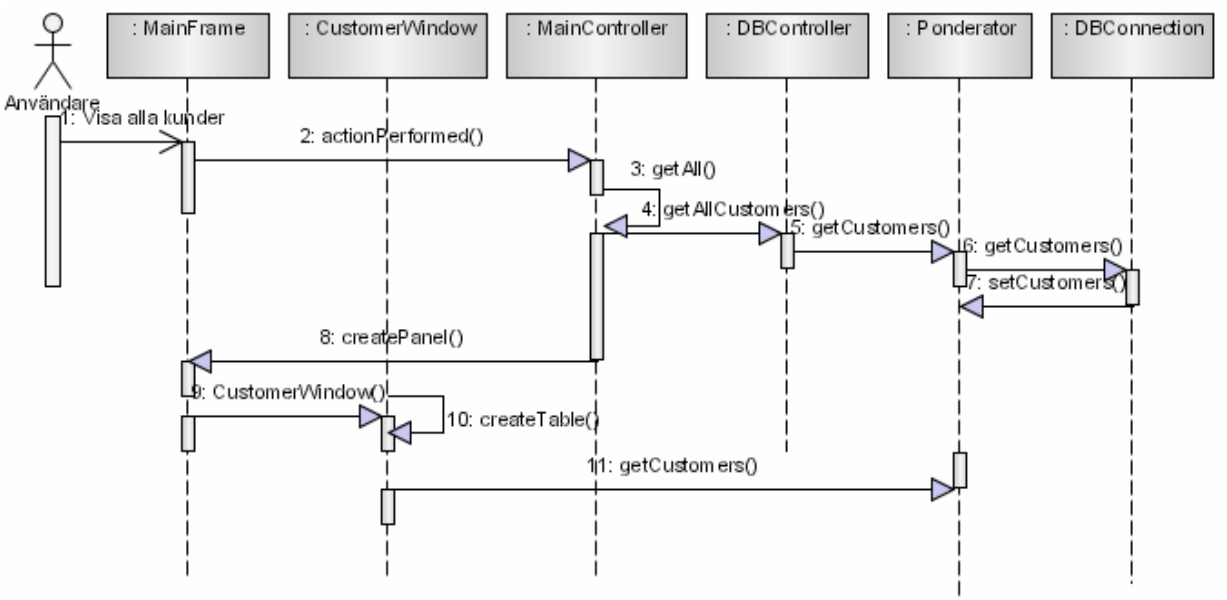
Användaren väljer här att redigera en befintlig offert genom att välja redigera i huvudfönstret (MainFrame) och sedan ”Offert” (se Figur 5.9). Ett popupfönster visas efter att händelsehanteraren (MainController) reagerat och användaren skriver in offertnumret. Den valda offerten visas genom att databaskontrollen vidarebefordrar önskemålet att öppna en viss offert till databasuppkopplingen. Kunden redigerar offerten och sparar.



Figur 5.9: Sekvensdiagram Visa-Redigera offert

5.5.4 Visa alla kunder

Sekvensdiagrammet visar hur användaren väljer att visa alla kunder i systemet i huvudfönstret genom Visa och ”Alla kunder” (se Figur 5.10). Händelsehanteraren (MainController) hämtar alla kunder genom att databaskontrollen vidarebefordrar önskemålet till databasuppkopplingsklassen (DBConnection) genom det interface som finns mellan controllern och modellen. Händelsehanteraren skapar sedan ett fönster för att visa alla kunder (CustomerWindow).



Figur 5.10: Sekvensdiagram Visa alla kunder

5.6 MS Excel-mallar

Företaget vill ha ett antal mallar för att skapa dokument kopior av offerter, order, ordererkännande och fakturor. Efter samtal med användarna och företaget föll valet av program på MS Excel. Användarna har tidigare erfarenhet av Excel och programmet kan erbjuda lite mer än vad MS Word kan i detta fall. Excel har funktioner för automatisk summering vilket passar mallarna mycket bra. Från Excel kan användarna infoga data direkt från databasen genom MS Query.

Eftersom företaget hade ett antal tidigare fakturor kunde vi utgå från dessa då mallarna skapades. Utseendet på de tidigare fakturorna var det inget fel på så jag valde att strukturera dem lite mer och sedan infoga formler för uträkning av summa med och utan moms. Den information som ska finnas i mallarna presenteras i Tabell 5.1.

Information/Fält	Beskrivning
Företagets namn	Ponderator AB
Kontaktinformation för Ponderator AB	Adress, mobil och fast telefon, hemsida, Bankgironummer
Fakturanummer	
Fakturadatum	
Kundens namn	
Kundens adress	
Kundens referens	
Ponderators referens	
Betalningsvillkor	30 dagar Netto
Produktlista	Produktbeskrivning, antal, styckepris, summa
Nettosumma	
Momssats	25 %
Totalsumma	

Tabell 5.1: Fakturamallens innehåll

Mallarna fungerar på samma sätt vare sig det är en order, offert eller faktura, så nedan följer endast en beskrivning av designen för fakturan. Mallen ska fungera på följande sätt: Användaren öppnar en fakturamall i Excel och sparar om den med fakturanummer som namn. Användaren fyller sedan i kunduppgifter och datum i respektive fält. För att minska risken för

fel så förs produkter och priser in direkt från databasen. Detta görs med hjälp av MS Query. Användaren väljer en befintlig fråga i MS Query och skriver in fakturanummer. Användaren blir tillfrågad var data ska placeras och väljer produktlistan i mallen.

5.7 Databasen

Kärnan i datasystemet är databasen, här ska all information om kunder, produkter offerter och så vidare finnas. Databasen skapades utefter de krav som kunden hade vid kravinsamlingen. Nedan följer en beskrivning av designen för databasen.

5.7.1 Entiteter och Attribut

Databasen ska innehålla följande entiteter:

- Kund, innehåller information om en kund.
- Produkt, innehåller information om en av företagets produkter.
- Offert, innehåller information om en offert som företaget skapat för en kund.
- Order, innehåller information om en order som företaget tagit emot från en kund.
- Faktura, innehåller information om en faktura som är skapat utifrån en beställning för en viss kund.

Dessa entiteter identifierades vid analysen av kraven i avsnitt 3.1. Till varje entitet finns även ett antal attribut. Attributen till ovanstående entiteter visas i Tabell 5.2.

Entitet	Attribut
Kund	Kundnummer, Namn, Gatuadress, Postadress, Tele, Referens
Produkt	Produktnummer, Namn
Offert	Offertnummer, Datum, Status, Vår Referens, Er Referens
Order	Ordernummer, Datum, Status, Vår Referens, Er Referens
Faktura	Fakturanummer, Datum, Status, Vår Referens, Er Referens

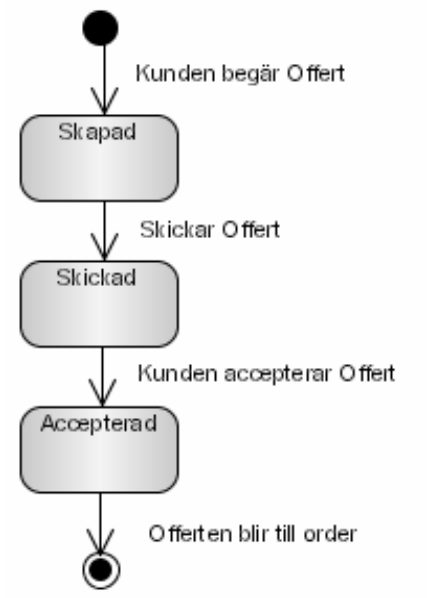
Tabell 5.2: Entiteter och attribut i databasen

5.7.2 Attributet ”Status”

Attributet status som finns i offert, order och faktura är till för att visa hur tillståndet för entiteten ser ut. Detta tillstånd är viktigt eftersom en offert inte får förändras efter att den skickas till exempel. Attributet kan illustreras med tillståndsdiagram för de olika entiteterna se Figur 5.11 - Figur 5.13.

Offert

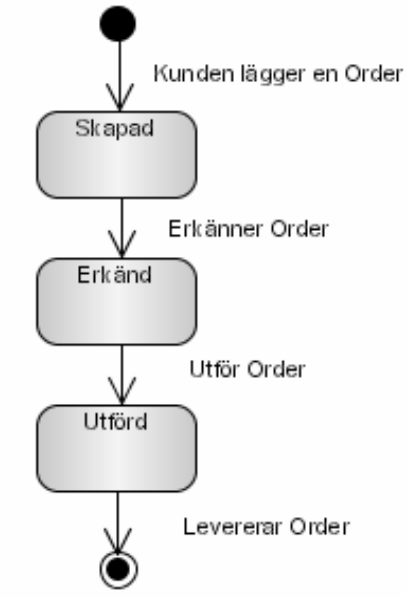
En offert skapas på begäran från en kund. Företaget skapar offerten och skickar den. Om kunden vill göra en beställning så accepteras offerten. Denna resulterar då i en order.



Figur 5.11: Tillståndsdiagram Status Offert

Order

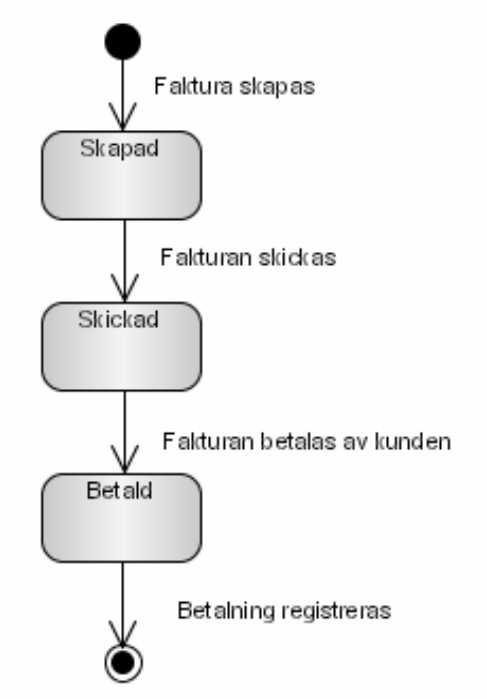
En order läggs av en kund. Företaget skapar då order och skickar ett ordererkännande. Företaget producerar ordern och levererar den.



Figur 5.12: Tillstådsdiagram Status Order

Faktura

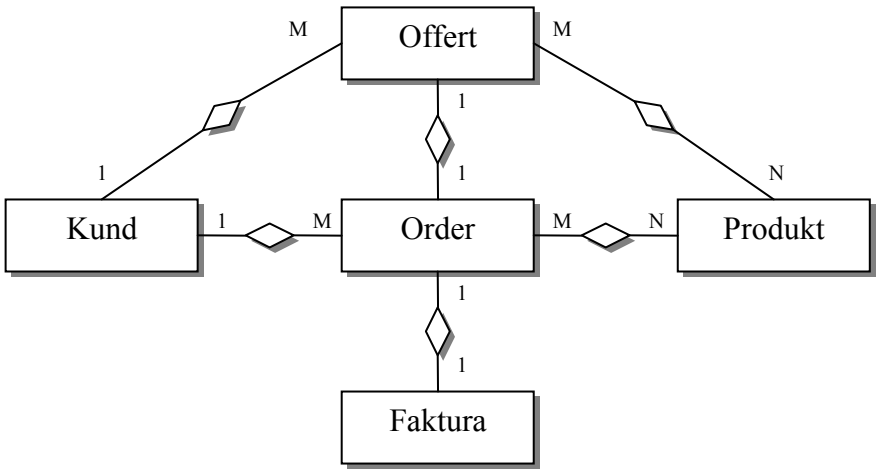
Företaget skapar en faktura från en levererad order. Företaget skickar fakturan. Kunden betalar fakturan och betalningen registreras.



Figur 5.13: Tillståndsdiagram Status Faktura

5.7.3 E-R diagram

Relationerna mellan entiteterna i databasen visas nedan i ett E-R diagram (entitet-relation). Dessa hjälper sedan till att skapa de tabeller som databasen ska innehålla. (Se Figur 5.14)



Figur 5.14: E-R diagram

Utifrån diagrammet ovan kan utläsas att en kund kan begära en offert eller göra en direkt beställning. Efter att en offert getts kan det ske en beställning och sedan skapas en faktura på den beställningen. Entiteten produkt innehåller produktnamn och produktnummer, de produkter som ska ingå i en offert, order eller faktura finns i relationen mellan dem.

5.7.4 Mappning från E-R-modellen till Relationsdatamodellen

För att skapa relationsdatabasen utifrån de entiteter och relationer som identifierats, mappas dessa till relationsdatamodellen [4].
Relationsdatabasen innehåller tabeller som motsvarar diagrammet ovan. Det tabeller som skapats är först och främst de entiteter som finns i diagrammet, samtidigt som varje många-till-många relation blir en egen tabell.

En till många relationen mellan kund och offert resulterar i två tabeller i relationsdatabasen där primärnyckeln från kund läggs in som främmandenyckel i offert. På samma sätt görs med kund – order relationen där kundnumret läggs in som främmandenyckel i order. Överförningen av primärnycklar görs även för relationerna Offert – Order och Order – Faktura.

Många – till många relationerna, Offert – Produkt och Order – Produkt, resulterar i egna tabeller där primärnycklarna från de ingående entiteterna skapar tabellens sammansatta primärnyckel. För Offert _ Produkt skapas tabellen Offert_prod som får primärnyckeln offertnr + produktnr och för Order – Produkt skapas tabellen orer_prod där primärnyckeln blir ordernr + produktnr. De tabeller som skapats för denna databas beskrivs nedan. Andra kolumnen i tabellen visar vilken datatyp varje attribut har.

Ett önskemål från kunden var att de inte skulle behöva skapa egna kundnummer, offertnummer och så vidare. Därför skapas dessa av databasen när en ny kund (eller någon annan entitet) läggs till. Kundnummer skapas från 100 upp till 999, offertnummer från 1000 till 2999, ordernummer från 3000 till 4999, fakturanummer från 5000 och upp till 9999. Produktnummer skapas från 10000 och uppåt eftersom produkterna är av största antalet.

5.7.5 Tabeller

Kund

Attribut	Datatyp	Värde	Övrigt
Kundnr	Integer (100-999)	NOT NULL	AUTO_INCREMENT
Namn	Varchar	NOT NULL	
Gatuadress	Varchar	NOT NULL	
Postadress	Varchar	NOT NULL	
Tele	Varchar		
Ref	Varchar		

Tabell 5.3:Tabell Kund i relationsdatabasen

Primärnyckel: Kundnr

Främmandenycklar: –

Tabellen har inga främmandenycklar eftersom den inte har några relationer som kräver mer än en kund.

Produkt

Attribut	Datatyp	Värde	Övrigt
Produktnr	Integer (10000 -)	NOT NULL	AUTO_INCREMENT
Namn	Varchar	NOT NULL	

Tabell 5.4: Tabell Produkt i relationsdatabasen

Primärnyckel: Produktnr

Främmandenycklar: –

Tabellen har inga främmandenycklar. Istället har nya tabeller skapas utifrån dess många-till-många förhållande med offert och order.

Offert

Attribut	Datatyp	Värde	Övrigt
Offertnr	Integer (1000-2999)	NOT NULL	AUTO_INCREMENT
Datum	Date	NOT NULL	
Status	Varchar		
ERef	Varchar		
VRef	Varchar		
Kundnr	Integer (100-999)	NOT NULL	

Tabell 5.5: Tabell Offert i relationsdatbasen

Primärnyckel: Offertnr

Främmandenycklar: Kundnr

Tabellen har Kundnr som främmandenyckel, efter en-till-många relationen med kund.

Order

Attribut	Datatyp	Värde	Övrigt
Ordertnr	Integer (3000-4999)	NOT NULL	AUTO_INCREMENT
Datum	Date	NOT NULL	
Status	Varchar		
ERef	Varchar		
VRef	Varchar		
Offertnr	Integer (1000-2999)		
Kundnr	Integer (100-999)	NOT NULL	

Tabell 5.6: Tabell Order i realationsdatabasen

Primärnyckel: Ordernr

Främmandenycklar: Kundnr , Offertnr

Tabellen har Kundnr och Offertnr som främmandenycklar, efter en-till-många relationen med kund och en-till-en relationen med order. En order kan göras med en offert som underlag eller direkt, därför finns dessa båda med.

Faktura

Attribut	Datatyp	Värde	Övrigt
Fakturanr	Integer (5000-9999)	NOT NULL	AUTO_INCREMENT
Datum	Date	NOT NULL	
Status	Varchar		
ERef	Varchar		
VRef	Varchar		
Ordernr	Integer (3000-4999)	NOT NULL	

Tabell 5.7: Tabell Faktura i relationsdatabasen

Primärnyckel: Fakturanr

Främmandenycklar: Ordernr

Tabellen har Ordernr som främmandenyckel, efter en-till-många relationen med order.

Offert_prod

Attribut	Datatyp	Värde	Övrigt
Offertnr	Integer (1000-2999)	NOT NULL	
Produktnr	Integer (10000 -)	NOT NULL	
Pris	Integer	NOT NULL	
Antal	Integer	NOT NULL	

Tabell 5.8: Tabell Offert_prod i relationsdatabasen

Primärnyckel: Offertnr + Produktnr

Främmandenycklar: -

Tabellen skapas ut många-till-många relationen mellan Offert och Produkt. Det är denna tabell som innehåller själva kroppen för en offert. Produktens pris och antal sätts också först här.

Order_prod

Attribut	Datatyp	Värde	Övrigt
Ordernr	Integer (3000-4999)	NOT NULL	
Produktnr	Integer (10000 -)	NOT NULL	
Pris	Integer	NOT NULL	
Antal	Integer	NOT NULL	

Tabell 5.9: Tabell Order_prod i relationsdatabasen

Primärnyckel: Ordernr + Produktnr

Främmandenycklar: -

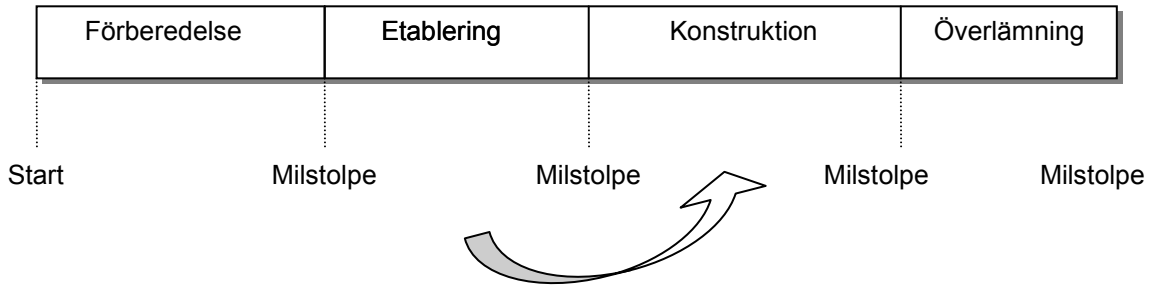
Tabellen skapas ut många-till-många relationen mellan Order och Produkt. Det är denna tabell som innehåller själva kroppen för en offert. Produktens pris och antal sätts också först här.

6 Implementationsmodellen & Implementation

I detta kapitel beskriver jag hur implementationen bryts ned till mer hanterbara komponenter. Kapitlet innehåller även en beskrivning av implementationsmodellens system. Sist i kapitlet beskrivs den färdiga implementationen av hela systemet uppdelat i vy, kontroll och modell.

6.1 Implementationsmodellen

Implementationsmodellens roll är att bygga vidare på designmodellens skelettstruktur av systemet, och delar in systemet i komponenter. Modellens komponenter är den källkod som skapas, skriptfiler, exekverbara filer och liknande. I detta skede sker även enhetstester av de komponenter som skapats, för att de ska kunna länkas ihop med andra komponenter till ett fungerande system.



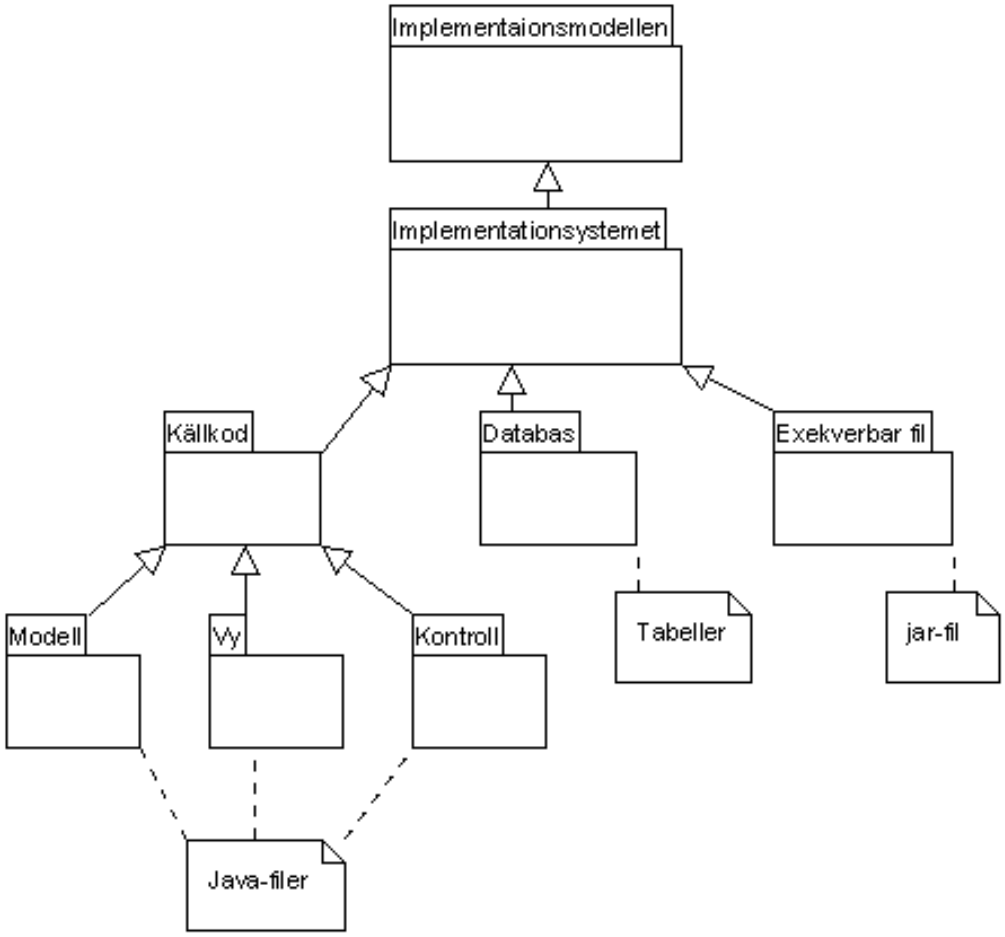
Figur 6.1: Nuvarande placering i utvecklingsprocessen

Implementationen sker till största delen under konstruktionsfasen (se Figur 6.1) och beskriver hur analysklasser från designmodellen ska implementeras ur komponentsynpunkt. Några av de artefakter som beskrivs i Implementationsmodellen är komponent, subsystem och interface.

Implementationsmodellen representeras av ett implementationsystem precis på samma sätt som designmodellen representeras av ett designsystem. Implementationsystemet motsvarar det högsta subsystemet i hierarkin.

6.2 Implementationssystemet

Implementationsmodellen resulterade i följande implementaionsystem i detta fall (se Figur 6.2)



Figur 6.2: Implementationssytemet

6.2.1 Subsystem

Subsystemen kan enkelt mappas till de subsystem som finns i designmodellen, där designklasserna nu blivit komponenter.

Inplementationsystemet består av subsystemen källkod, databas, exekverbar fil, vy, modell och kontroll. De tre förstnämnda innehåller endast subsystem och de tre sistnämnda innehåller komponenter (se Tabell 6.1). Komponenterna är de faktiska javafilerna som har implementerats, dessa har en motsvarande designklass i designmodellen.

Subsystemen vy, modell och kontroll har naturliga kopplingar genom designprincipen beskriven i avsnitt 5.3.1. Kontrollen är kopplad och beroende av både modellen och vyn,

medan vyn endast representerar modellen grafiskt. Det gränssnitt som används för att koppla kontrollen till modellen finns i modellen subsystem och skickas med då modellen skapar kontrollen och vyn. Ytterligare beskrivning av funktionaliteten finns i avsnitt 6.4

Subsystem	Innehåll (subsystem/komponenter)
Källkod	Vy, Modell, Kontroll
Databas	Tabeller
Exekverbar fil	jar-fil
Vy	.java filer för användargränssnitt
Modell	.java filer för modellen
Kontroll	.java filer för hantering av händelser och databas

Tabell 6.1: Subsystem och komponenter i implementationsystemet

6.3 Implementation av grafiskt användargränssnitt

Eftersom företaget vill ha ett grafiskt gränssnitt så skapade jag de grafiska komponenterna först. Genom att skapa gränssnittet först får kunden en bild av hur systemet kommer att se ut och jag kan tidigt göra förändringar om det skulle behövas. Kunden får också en känsla för hur systemet kommer att fungera genom att se en bild av det och samtidigt av mig få en förklaring av funktionaliteten i stora drag. Enligt [5] är det viktigt att snabbt få med användaren i utvecklingen för att ta del av deras kunskap samt att i ett tidigt skede vänja användaren vid gränssnittet.

Användargränssnittet är uppbyggt av ett grundfönster som innehåller två olika paneler. Ena panelen innehåller textfält och tabeller med information och den andra knappar för att spara och ändra informationen. På detta sätt är det enkelt att byta ut panelen som innehåller information till de olika typer som ska representeras (se Figur 5.4).

6.3.1 Vyn

Jag har valt att bygga upp datasystemet med MVC–modellen (se avsnitt 5.3.1), användargränssnittet blir då vyn i modellen. Användargränssnittet ska endast visa information samt vidarebefordra den information som användaren skriver in. Inget ska lagras utan all kommunikation med databasen sker genom kontrollklasserna och det definierade gränssnittet mot modellen. I denna implementationen av vyn finns följande klasser.

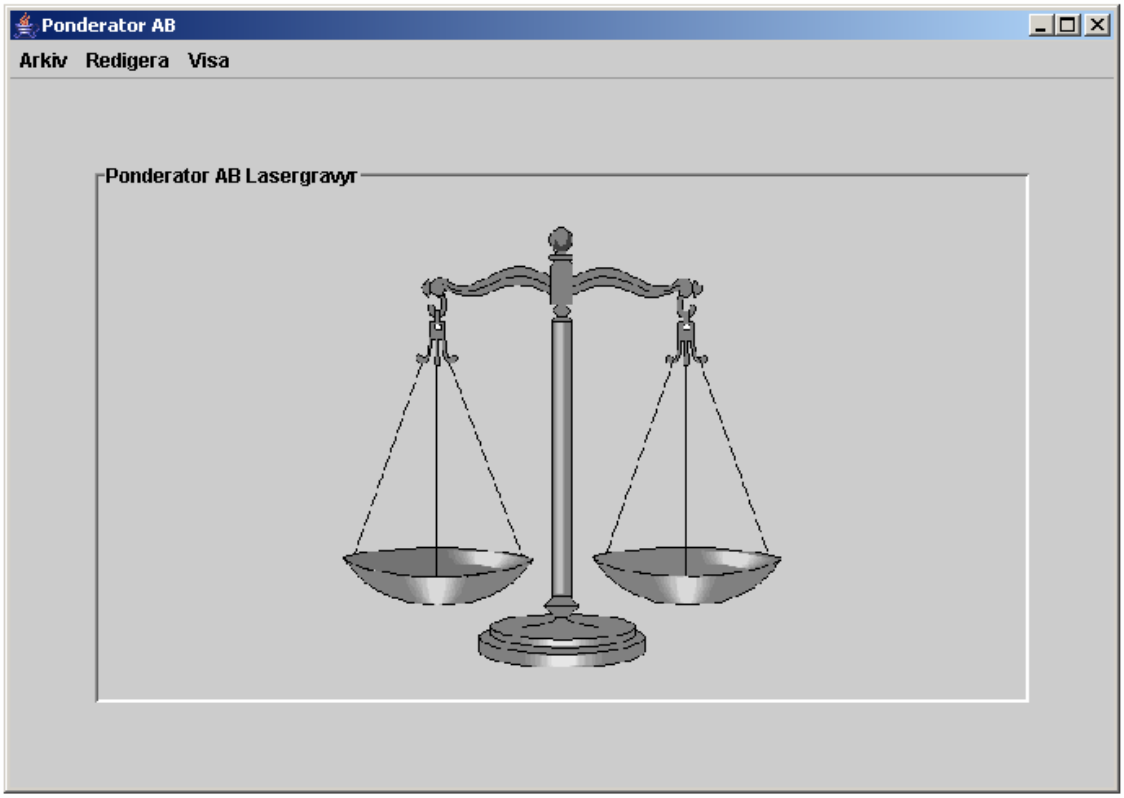
MainFrame – Huvudfönster och startfönster som innehåller alla grafiska komponenter. Innehåller från början endast menyerna ”Arkiv”, ”Redigera” och ”Visa”. I dessa menyer finns följande alternativ

Arkiv – ”Ny” (Kund, Offert, Order, Faktura) och ”Avsluta”.

Redigera - ”Kund”, ”Offert”, ”Order”, ”Faktura” ”Produkt”(Lägg Till och Ta Bort).

Visa – ”Alla Kunder”, ”Alla Offerter”, ”Alla Order”, ”Alla Fakturor”, ”Alla Produkter”

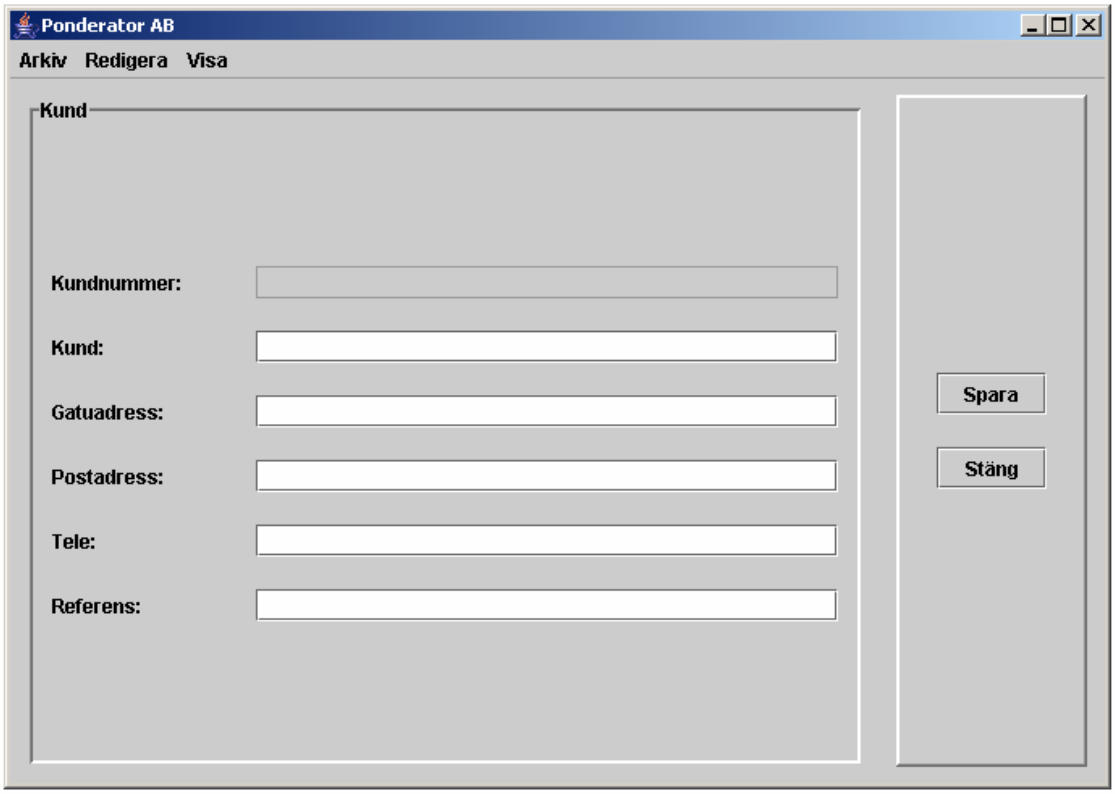
Användaren navigerar härifrån till de olika vyerna för att lägga till information i databasen.



Figur 6.3: Huvudfönster (Ponderator betyder vågmästare)

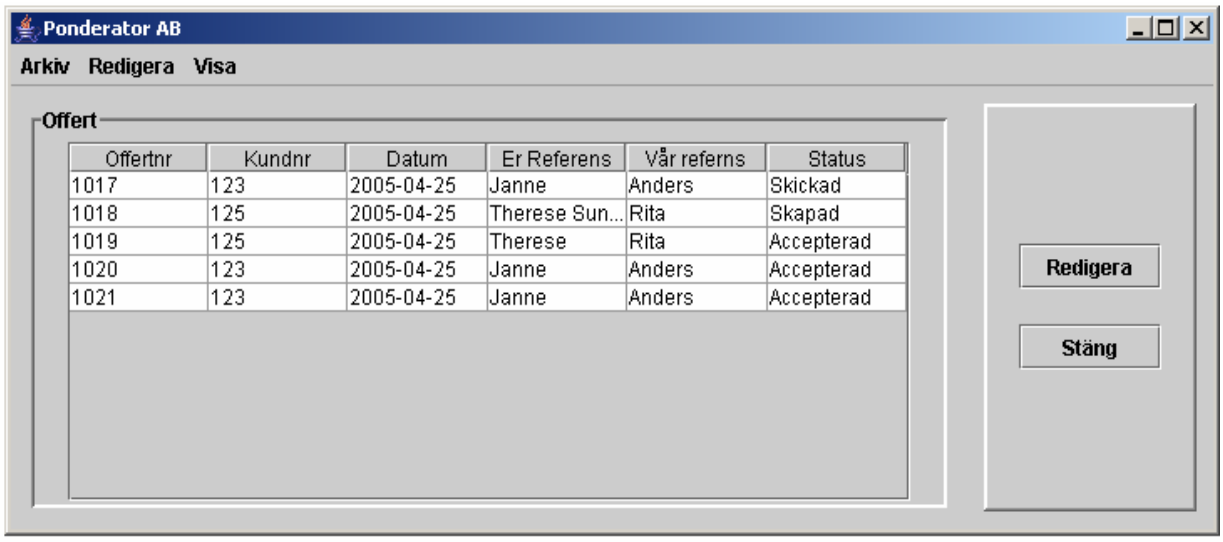
InformationPanel – Del av huvudfönstret som innehåller textfält och tabeller beroende på vad det är användaren vill göra. Denna klass innehåller en panel som är någon av typerna CustomerWindow, OfferWindow, OrderWindow, InvoiceWindow, ProductWindow (se Figur 6.4 - Figur 6.6). Panelen har inte några egna grafiska komponenter som händelsehanteras utan dessa finns i panelens underliggande panel

CustomerWindow – Innehåller grafiska komponenter för att kunna skapa och redigera kundinformation (se Figur 6.4). Huvudfönstret innehåller då även en panel med knappar Buttons där knapparna ”Spara” och ”Stäng” visas. Fönstret har även möjligheten att visa alla kunder precis som ”OfferWindow” (se Figur 6.5)
Kan välja att skapa en ny kund i Arkivmenyn och då visas detta fönster med blanka textfält. Kundnummer fältet är inte editerbart eftersom kundnumret skapas av databasen. Användaren skriver in kundinformation och sparar genom knapp-menyn till vänster. Fönstret kan också öppnas genom Redigera-menyn, eller genom att först visa alla kunder och sedan välja att redigera en specifik kund. Fönstret visas då med ifyllda fält och användaren kan välja att ändra informationen eller bara läsa av den.



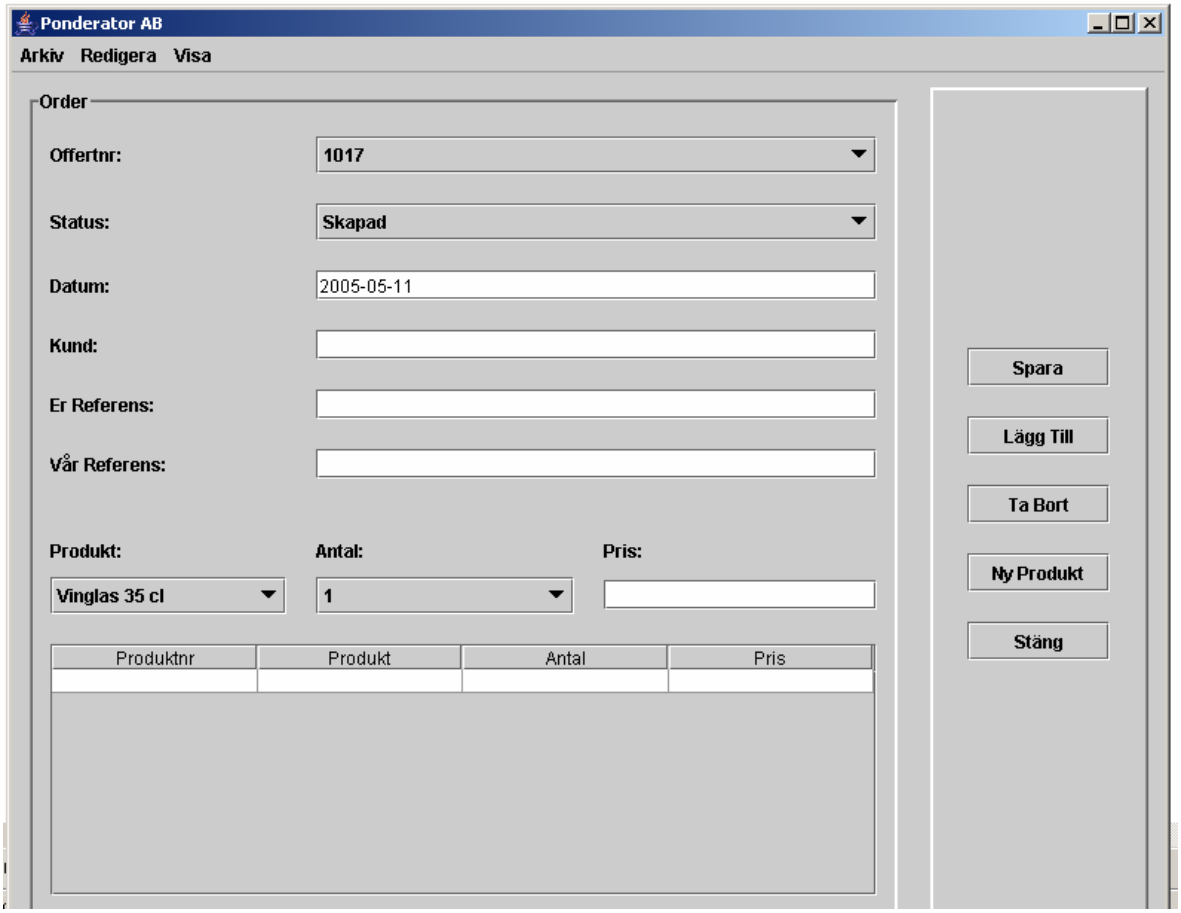
Figur 6.4: Fönster för att skapa en ny kund.

OfferWindow – Innehåller grafiska komponenter för att kunna spara och redigera offertinformation. Panelen med knappar visar här redigera och stäng eftersom en av offerterna kan väljas för redigering (se Figur 6.5). Fönstret kan vivas genom Arkiv och Ny offert. Då kommer alla fält att vara tomma utom datum- och statusfältet. Användaren väljer då att fylla i information på samma sätt som OrderWindow nedan.



Figur 6.5: Klassen OfferWindow visar alla offerter

OrderWindow – Innehåller grafiska komponenter för att kunna spara och redigera orderinformation. Fönstret visas genom Arkiv och ny Order. Fönstret ser då ut som i Figur 6.6. När ett offertnummer väljs i listan så visas automatiskt den offertens kund och referensinformation. Samtidigt som kundinformationen fylls i så visas ett mindre fönster med de produkter som finns i offerten med det valda offertnumret. Användaren väljer då att infoga alla produkter eller markerar de som ska med till ordern. Produkter kan även läggas till genom att välja en produkt i listan och välja antal och pris. Genom att användaren sedan trycker på Lägg Till knappen så hanar produktinformationen i tabellen. Om användaren behöver lägga till en ny produkt så används knappen Ny Produkt. Ett litet popupfönster visas och produktnamnet kan skrivas in. Listan med produkter i offertfönstret uppdateras så att användaren kan välja produkten.



Figur 6.6: Klassen OrderWindow med formulär för att skapa en order

InvoiceWindow – Innehåller grafiska komponenter för att kunna spara och redigera fakturainformation. Innehåller samma komponenter som Offer och OrderWindow, fast anpassat till faktura så att användaren väljer ordernummer istället för offertnummer.

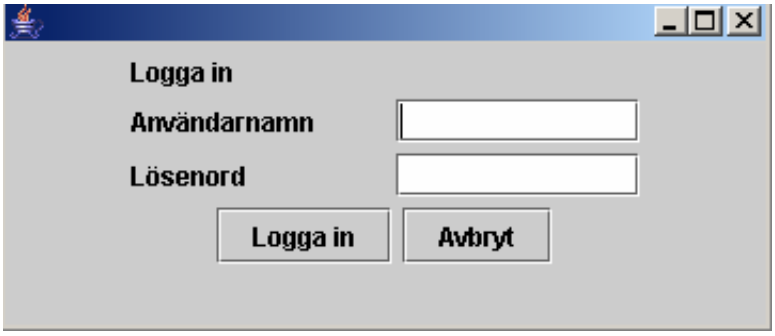
ProductWindow – Innehåller grafiska komponenter för att kunna visa alla produkter på samma sätt som OfferWindow (se Figur 6.5). I detta fönster finns även möjligheten att ta bort en viss produkt.

ProductsFromOffer – Innehåller en tabell för att visa de produkter som finns i en viss offert eller order för att användaren ska kunna välja vilka produkter som ska tas med i ordern eller fakturan. Visas i huvudfönstret tillsammans med en panel för knapparna ”Markerade” och ”Alla”

Buttons – Den del av huvudfönstret som innehåller knappar för att spara och redigera information. Följande knappar finns:

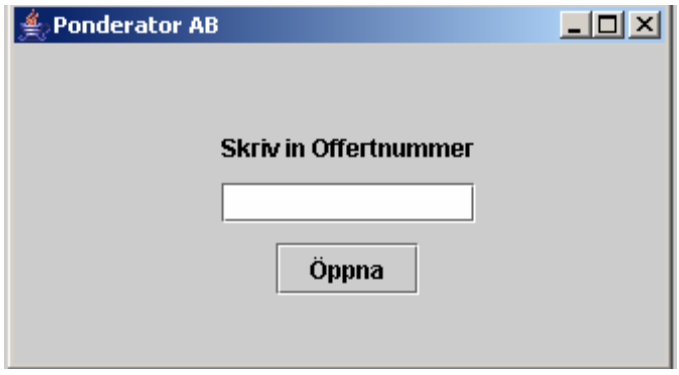
Spara – sparar innehållet i panelen till vänster (Kund, offert, order eller faktura).
Ändra – Ändrar det öppnade objektet i databasen (när användaren först valt redigera)
Lägg till – Lägger till en produkt i produkttabellen för en offert, order eller faktura.
Ta Bort – Tar bort en produkt ur tabellen för en offert, order eller faktura.
Ny Produkt – Skapar ett fönster för att användaren ska kunna skapa en ny produkt.
Redigera – Öppnar den valda raden i tabellen för redigering.
Stäng – Stänger aktuellt fönster.
(Knapparna kan ses i Figur 6.4 - Figur 6.6)

PopUpLogIn – Popupfönster med grafiska komponenter för att användaren ska kunna logga in till systemet. Om användarnamn eller lösenord är felaktigt visas ett felmeddelande och användaren uppmanas att försöka igen.



Figur 6.7: Logga in ruta ”PopUpLogIn”

PopUpFrame – Popupfönster med grafiska komponenter för att kunna plocka upp en viss kund, offert, order eller faktura.



Figur 6.8: PopUpfönster för att hämta kund, offert, order eller faktura.

PopUpProduct – Popupfönster med grafiska komponenter för att kunna lägga till eller ta bort en produkt ur databasen ser ut på samma sätt som PopUpFrame (se Figur 6.8).

6.4 Implementation av modell och kontroll

Designen av systemet bygger som beskrivet i avsnitt 5.3.1 på Model-View-Controller modellen. Systemets funktionalitet ligger till största delen i kontrollpaketet i modellen. (se Figur 5.3) Jag kommer i detta avsnitt beskriva hur funktionaliteten i systemet är uppbyggt och hur kommunikationen med databasen ser ut.

6.4.1 Kontroll

Kontrollpaketet innehåller två kontrollklasser samt de hjälpklasser som hela systemet kan använda sig av. Kontrollen har till uppgift att reagera på de händelser som sker genom användarens interaktion med gränssnittet samt att skapa de frågor som överförs till databasen.

Följande klasser finns i kontrollpaketet:
MainController – Hanterar händelser från gränssnittsklasserna och utför motsvarande anrop mot databasen eller tillbaka till gränssnittet.
DBController- Hanterar skapandet av frågor till databasen och utför även kontroller på de objekt som ska till databasen.
Customer – Hjälpklass för att lagra en kund som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

Offer- Hjälpklass för att lagra en Offert som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

Order- Hjälpklass för att lagra en Order som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

Invoice- Hjälpklass för att lagra en Faktura som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

Product - Hjälpklass för att lagra en Produkt som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

OfferProduct - Hjälpklass för att lagra en OffertProdukt (en produkt som tillhör en viss offert se avsnitt 5.7.4) som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

OrderProduct- Hjälpklass för att lagra en OrderProdukt (en produkt som tillhör en viss order se avsnitt 5.7.4) som hämtats eller ska lagras i databasen. Används av hela systemet som temporär lagring av data. Innehåller enbart strängrepresentationer av data i databasen.

6.4.2 Modell

Modellen innehåller de delar av systemet som utgör statisk information, till exempel databasen. I detta fall finns även databasuppkopplingklassen i modellen samt systemets lagringsplats för aktiva objekt. Nedan följer en beskrivning av modellpaketets klasser:

DBConnection- Innehåller databasuppkopplingen. Hanterar inloggning samt exekvering av sql-frågor mot databasen. Vid hämtning av data från databasen lagras resultatet om till respektive objekt och sätts i systemets lagringsplats för aktiva objekt (se Ponderator).

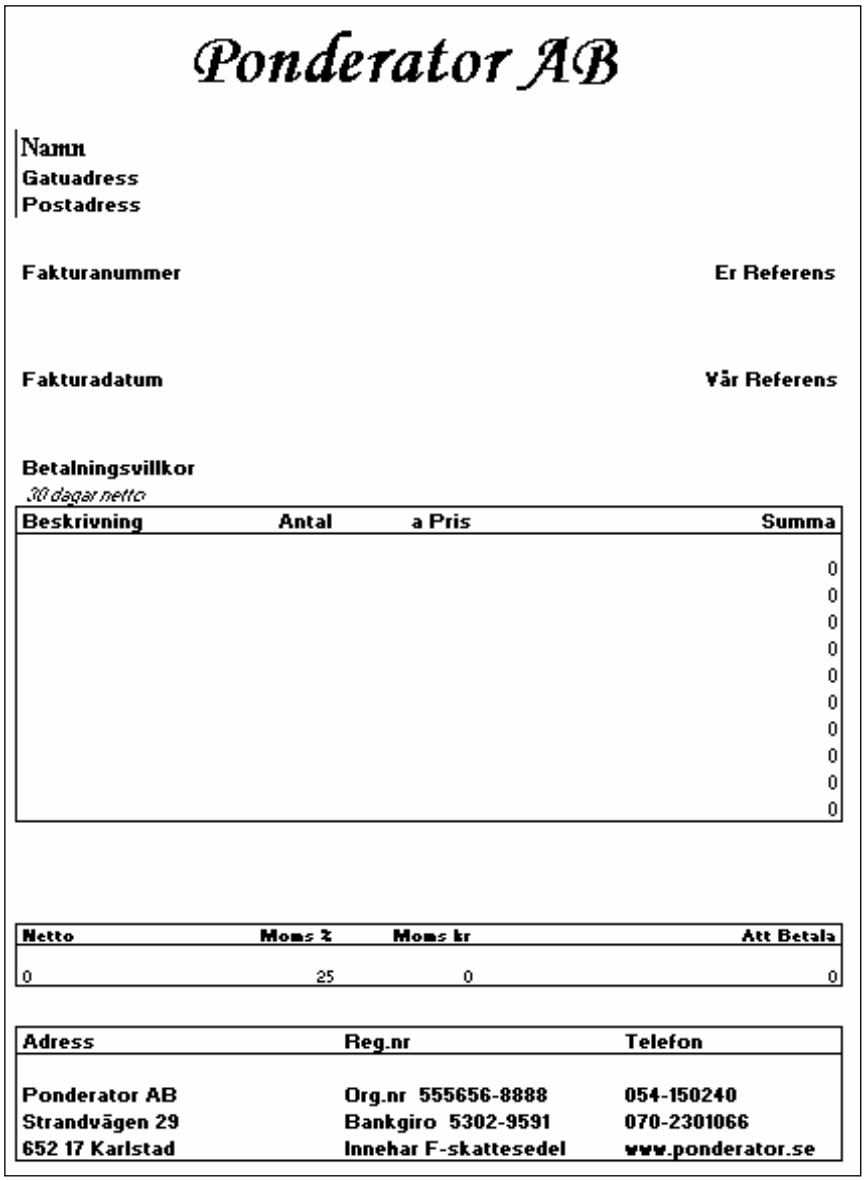
Ponderator- Innehåller en lagringsplats för systemets aktiva objekt, till exempel aktiv kund, order eller offert. Lagringen av objekt på detta sätt gör att objekt inte behöver skickas med vid anrop, den klass som är intresserad av objektet får fråga efter det istället. Klassen implementerar Model_Interface som tillåter kontrollen och vyn att få tillgång till systemets aktiva objekt samt att hämta och lagra information i databasen.

Model_Interface- Gränssnittet innehåller metoder för att sätta och hämta systemets aktiva objekt samt metoder för att sätta och hämta information från databasen. Kontrollen och vyn tar emot sina modellobjekt som detta interface och kan därför endast komma åt de metoder som finns definierade i interfacet.

StartUp- Agerar startklass för systemet.

6.5 MS Excel-mall och frågor

Mallarna för offerter, order, och fakturor skapades i MS Excel. Alla mallar har samma utseende och funktion och därför beskriver jag endast hur jag skapade fakturamallen.
Mallen innehåller att antal fält som användaren fyller i. Besparingen som mallen ger är att användaren inte behöver formatera mallen varje gång och kan föra in långa produktlistor direkt från databasen. Mallen summerar också fakturan med och utan moms (se Figur 6.9).


Ponderator AB

Namn
Gatuadress
Postadress

Fakturanummer — Er Referens

Fakturadatum — Vår Referens

Betalningsvillkor
30 dagar netto

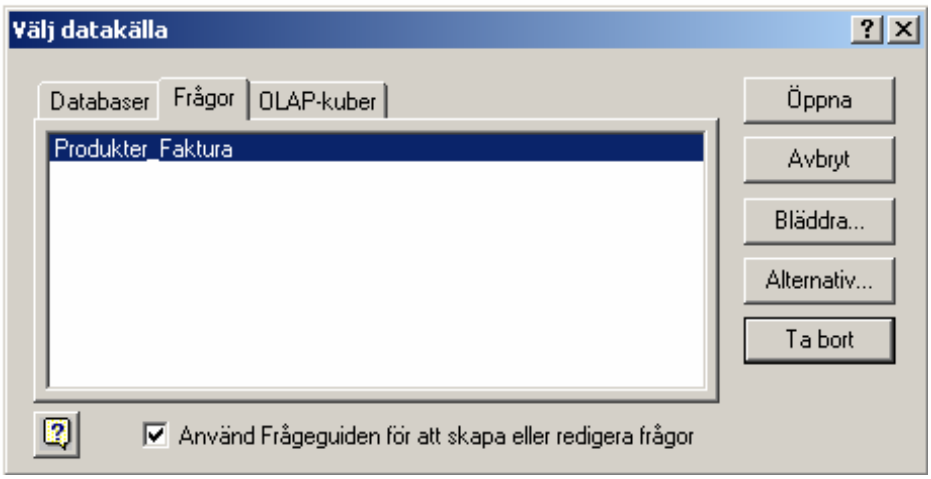
Beskrivning	Antal	a Pris	Summa
			0
			0
			0
			0
			0
			0
			0
			0
			0
			0

Netto	Moms %	Moms kr	Att Betala
0	25	0	0

Adress	Reg.nr	Telefon
Ponderator AB	Org.nr 555656-8888	054-150240
Strandvägen 29	Bankgiro 5302-9591	070-2301066
652 17 Karlstad	Innehar F-skattesedel	www.ponderator.se

Figur 6.9: Fakturamall

Produkterna förs sedan in genom MS Query. Jag har skapat en fråga som går att redigera beroende på vilken faktura produkterna ska tas från. Användaren behöver bara välja Data -> Infoga Externa Data -> Ny Databasfråga och i fönstret välja Frågor (se Figur 6.10).



Figur 6.10: Val av fråga i MS Excel

Användaren väljer fråga och kan redigera frågan i MS Query genom att ange det ordernummer som fakturan beror på. (se avsnitt 5.7). Ms Query hämtar de produkter som finns och visar dessa i ett fönster. Användaren väljer Arkiv -> Returnera data till Excel.
Programmet frågar så användaren vart data ska placeras och användaren väljer produktlistan i mallen.

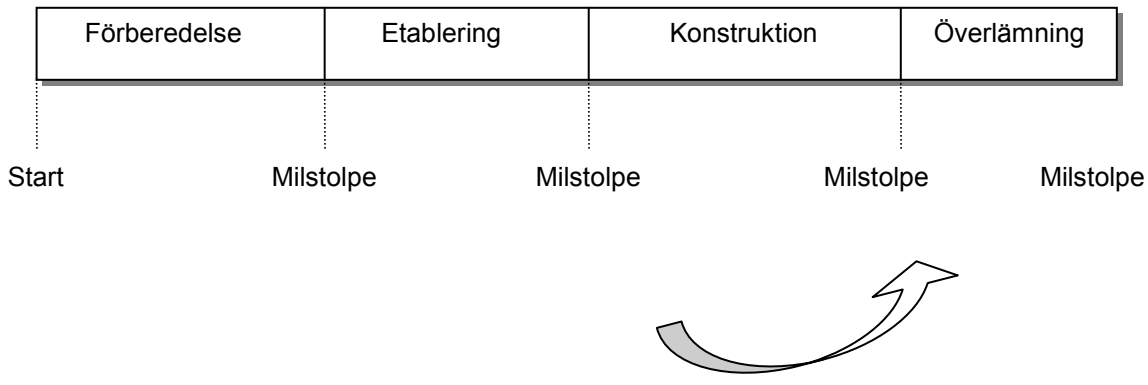
7 Testmodellen & Överlämning

Detta kapitel innehåller en beskrivning av hur systemet har testats. Testmodellen bygger precis som design- och implementationsmodellen på ett system (testsystemet) som också beskrivs i kapitlet. Ett antal testfall beskrivs senare i kapitlet som kan mappas till användningsfallen i avsnitt 3.4.2. Sist i kapitlet beskriver jag överlämningen av systemet till företaget.

7.1 Inledning

Testmodellen används för att specificera de tester som utförs på olika komponenter och på hela systemet. Testerna utförs under hela utvecklingsprocessen eftersom utvecklingen sker iterativt. Modellen innehåller testfall (*test case*) som verifierar användningsfallen i användningsfallsmodellen och kan även innehålla testfall för vare användningsfallsrealisering i design och implementationsmodellen.

Enligt [9] motsvarar verifieringen av användningsfallen ett ”black-box”-test, och verifiering av realiseringen av användningsfallen ett ”white-box”-test. ”Black-box”-testet kontrollerar hur systemet reagerar på utsidan medan ”white-box”-testet kontrollerar funktionen mellan delar av det interna systemet.



Figur 7.1: Nuvarande placering i utvecklingsprocessen

7.2 Testsystemet

Testmodellen innehåller precis som de tidigare modellerna ett system för att modellera alla ingående delar. I testmodellen finns testsystemet som i sin tur innehåller testfall, testprocedur och testkomponent. Testfallet och testproceduren är nära sammankopplade medan testkomponenten innehåller ett automatiserat test som motsvarar hela eller en del av testfallet. Testkomponenten är inte alltid möjlig och inte heller alltid nödvändig.

7.2.1 Testfall och testprocedur

Under utvecklingen av systemet i detta fall har jag skapat testfall för att verifiera användningsfallen. Systemet bygger på ett grafiskt användargränssnitt och en databas och interaktionen mellan dessa. Testfallen ska naturligtvis verifiera de för- och eftervillkor som satts upp för varje användningsfall. Flera testfall kan finnas för ett och samma användningsfall eftersom olika scenarion kan uppstå, till exempel att kundnamnet redan existerar i systemet men användaren lägger till det ändå. I varje testfall specificeras indata, resultat och övriga krav. Testfallet beskrivs sedan utförligare steg för steg i en testprocedur (*test procedure*). Även automatiserade tester av komponenter kan finnas i testmodellen och även en utvärdering av resultaten om många tester utförts.

Testproceduren talar om för den som ska utföra testet hur det ska utföras. Proceduren är uppbyggt av ett antal steg som ingående beskriver vad som ska matas in, vilka knappar och menyer som ska användas. I detta fall har jag specificerat testfallen lite mer utförligt istället för att skapa en procedur för varje testfall, eftersom enbart ett fåtal komponenter berörs vid varje testfall.

7.3 Specifikation av testfall

Nedan följer en specifikation av några av de testfall som jag har skapat. Vissa av testfallen inkluderar delar från tidigare eller senare tester och refererar då till dessa. Dessa testfall verifierar slutprodukten, och är inte testfall för någon av de produkter som skapats under en iteration.

Ny_kund_ok

Indata: Kundinformation bestående av

Namn: Företag 1

Gatuadress: Gatan 1

Postadress: Staden 2

Tele: 012-345678

Referens: Anders Karlsson

Skriv in informationen i respektive fält i kundformuläret och tryck på spara.

Resultat: Kunden sparas i databasen och får ett unikt kundnummer mellan 100 och 999. Kunden finns längst ned i listan då alla kunder visas.

Ny_kund_existerar

Indata: Databasen innehåller kunden med kundinformation bestående av:

Namn: Företag 1

Gatuadress: Gatan 1

Postadress: Staden 2

Tele: 012-345678

Referens: Anders Karlsson

Skriv in informationen i respektive fält i kundformuläret och tryck på spara.

Resultat: Ett felmeddelande visas eftersom kunden redan finns i systemet. Användaren uppmanas att använda ett annat namn. Databasen förblir oförändrad.

Ny_kund_IngenAdress

Indata: Databasen innehåller ingen kund med kundnamnet i kundinformationen bestående av:

Namn: Företag 2

Gatuadress:

Postadress: Staden 2

Tele: 654-345678

Referens: Skriv in informationen i respektive fält i kundformuläret och tryck på spara.

Resultat: Ett felmeddelande visas eftersom Gatuadressen inte är ifylld. Användaren uppmanas att fylla i alla obligatoriska fält. Databasen förblir oförändrad.

Visa/Redigera_Offert

Indata: Databasen innehåller en offert med offertnummer 1000 och status ”Skapad”
Välj Redigera Offert och skriv in 1000, tryck på ”öppna”.

Resultat: Offerten med offernummer 1000 visas i fönstret. Knappanelen innehåller knappar för att redigera offerten samt ”stäng”.

Visa/Redigera_Offert_Skickad

Indata: Databasen innehåller en offert med offertnummer 1000 och status ”Skickad”
Välj Redigera Offert och skriv in 1000, tryck på ”Öppna”.

Resultat: Ett meddelande visas på skärmen och talar om att offerten är skickad och att endast statusen kan ändras.
Offerten med offernummer 1000 visas i fönstret. Knappanelen innehåller knapparna ”Ändra” offerten samt ”Stäng ”.

Visa/Redigera_Offert_Lista_Accepterad

Indata: Databasen innehåller en offert med offertnummer 1000 och status ”Accepterad”
Välj Visa Alla offerter och välj offert med nummer 1000, tryck på ”Redigera”.

Resultat: Ett meddelande visas på skärmen som talar om att offerten är accepterad och kan endast visas.
Offerten med offernummer 1000 visas i fönstret. Knappanelen innehåller knappen ”Stäng”.

Visa_Alla_Kunder

Indata: Databasen innehåller ett antal kunder.
Välj Visa Alla kunder.

Resultat: En lista med alla kunder i systemet visas, knappanelen innehåller ”Redigera” och ”Stäng”.

Ny_produkt_ok

Indata: Databasen innehåller inte någon produkt med produktinformation bestående av

Namn: Vinglas 30 cl

Skriv in informationen i fältet i popupfönstret och tryck på ”Lägg Till”.

Resultat: Produkten sparas i databasen och får ett unikt produktnummer med startnummer 10000. Produkten finns längst ned i listan då alla produkter visas.

Ny_produkt_existerar

Indata: Databasen innehåller produkten med produktinformation bestående av

Namn: Vinglas 30 cl

Skriv in informationen i fältet i popupfönstret och tryck på ”Lägg Till”.

Resultat: Ett meddelande visas på skärmen som talar om att produkten redan finns. Användaren uppmanas att välja ett annat namn. Databasen påverkas inte.

7.4 Överlämning

Överlämning av systemet görs då det har bedömts att fungera stabilt i den tänkta användningsmiljön. Systemet behöver inte vara perfekt vid överlämning, systemet kan påvisa defekter då det används i sin rätta miljö. Detta är just vad överlämningsfasen är till för. Fasen ska se till att alla de krav som finns på systemet ska ha uppnåtts, samt se till att systemet testas i sin rätta miljö.

Jag har under denna fas installerat systemet på Ponderator AB. Installationen krävde nedladdning av MySql Server och Java JRE (*Java Runtime Environment*). Programmet startas med hjälp av en jar-fil. Överlämningen innehöll även en detaljerad instruktion till användarna om hur systemet fungerar. Jag har i överlämningsfasen även skapat en lätthanterlig användarmanual för systemet, MS Excel-mallarna och MS Query.

Artefakter som överlämnats är:

- Användarmanual: System
- Användarmanual: Excel/MS Query
- Körbar jar-fil

I det normala fallet används överlämningsfasen för att testa systemet i dess naturliga miljö. Jag har testat systemet till viss del i dess riktiga miljö. Eftersom miljön inte skiljer sig så mycket från den miljö som systemet utvecklades i använder nu företaget systemet under en period för att se om det behöver utvecklas vidare.

8 Resultat & Rekommendationer

Min uppgift var att skapa ett affärssystem åt företaget Ponderator AB. Företaget ville ha ett system där de kunde skapa och hantera offerter, order, fakturor, produkter och kunder. De ville också ha en mall för att skapa dokument av offerter, order och fakturor i MS Excel.
Resultatet av projektet blev en databas som lagrar informationen som företaget önskade. Kopplingen till databasen görs genom ett gränssnitt implementerat i Java, och överföringen av information mellan databasen och gränssnittet görs med JDBC.

8.1 Resultat av projektet med UP

Projektet jag gjort har utförts enligt Unified Process och jag har genom hela arbetet teoretiskt beskrivit stegen i metoden. Jag valde att använda denna metod eftersom den innefattar så mycket samtidigt som den är anpassningsbar till varje specifikt projekt. När arbetet startade var inte företaget och användarna helt säkra på vad de ville ha och vad de behövde. Eftersom UP använder sig i stor utsträckning av kravinsamling, kravanalys och där till användningsfall tyckte jag att metoden passade bra i detta fall.

Efter ett flertal intervjuer med användarna skapade jag användningsfallsdiagram och beskrivningar till dessa som jag gick igenom med tillsammans med dem. Efter att jag strukturerat kraven och fått reda på vad företaget faktiskt var i behov av kunde det praktiska arbetet fortlöpa utan några större problem. Alla de delar som företaget hade önskemål om har skapats genom databasen, användargränssnittet och mallar för offerter, order och fakturor i MS Excel. Arbetet utfördes i nära samarbete med kunden och användarna och de är mycket nöjda med produkten. Systemet är i drift och företaget kommer att använda sommaren som en försöks- och utvärderingsperiod för systemet för att se om det behöver förändras eller utökas.

Under arbetets gång har jag dokumenterat varje steg i Unified Process och därigenom också skapat ett antal modeller. Dessa modeller sträcker sig från användningsfall som illustrerar kundens krav, till testfall som verifierar funktionaliteten implicerad av användningsfallen. UP är en mycket omfattande metod och är svår att få en bra översiktsbild av. Jag tycker ändå att

de modeller som jag har beskrivit i uppsatsen ger en bra insyn i hur metoden fungerar och påverkar utvecklingen.
Utvecklingen med UP går igenom fyra faser (se avsnitt 2.6.1), som alla utförs i iterationer. Inledningsfasen (*inception*) var mycket kort i detta fall eftersom det från början stod klart att detta projekt skulle kunna genomföras. Fasen innehöll endast en iteration. Etableringsfasen (*elaboration*) fick mer tid eftersom kravinsamling och kravanalys skedde i denna fas. Trots att fasen innehöll mycket arbete krävdes endast två iterationer. Konstruktionsfasen (*construction*) skedde i fler iterationer eftersom användargränssnittet skapades tidigt så att användaren kunde få en bild av systemet och därmed också tidigt kunna göra förändringar i kraven. Konstruktionsfasen utfördes genom fyra iterationer. Överlämningsfasen som egentligen innefattar tester av systemet på plats, samt utbildning av systemets användare krävde endast en iteration. I denna fas har jag lagt skapandet av mallarna samt installation av produkten.
Eftersom arbetet har fortlöpt utan några större problem har antalet planerade iterationer stämt bra överens med det faktiska antalet iterationer.

UP lämnar stort utrymme för egna designprinciper och lägger stor vikt på arkitektur. Jag har i detta projekt använt MVC-modellen för att ge systemet struktur och lägga en god grund för utbyggbarhet [10], och det har fungerat mycket bra. Den koppling som har behövts mellan Java och mySql samt mellan MS Excel och mySql har varit smidig tack vare bra dokumentation på mySql.com.

De problem som uppstått under projektet har varit ganska små och lätta att lösa. Det största problemet är ändå att skapa ett system utifrån krav och önskemål från en kund. Implementationsmässigt hade jag problem då en ny offert eller order skulle skapas. När en ny offert ska lagras i databasen ska dess produkter finnas i tabellen offert_prod. Eftersom databasen skapar de unika offert- och ordernumren så kunde jag inte få tag på det nummer som skulle lagras i offert_prod tabellen. Detta beror på att databasen skapar numret och eftersom det inte fanns tidigare hade jag inget att söka på. Lösningen var att leta upp det största numret i tabellen eftersom det unika numret alltid ökar med ett. Offerter tas aldrig bort och därför finns heller inga hål i den serien av nummer. På samma sätt gjorde jag med order och order_prod.

8.2 Rekommendationer

Systemet som jag skapat innehåller inte så många olika delar som ska samverka. Därför har jag valt att göra en mycket enkel design. I flertalet fall har jag funderat på att använda olika designmönster [6], så som Observer för automatiska uppdateringar från databasen, Composite eller Abstract Factory för uppbyggnad av användargränssnittet. Jag lade istället ned mycket tid på att infria företagets krav (och i vissa fall endast önskemål) om systemets utseende, funktionalitet och användarvänlighet. Om systemet skulle utökas eller förändras tycker jag att Observer-mönstret ska läggas till och kanske även Adapter för att föra information från vyn till modellen. Observer fungerar mycket bra ihop med MVC-modellen och Java har inbyggt stöd för mönstret.

Ytterligare funktionalitet kan även läggas till, till exempel sökning med fler parametrar och utskriftsmöjligheter direkt från programmet. Även funktionalitet för att kunna öppna en ny offert, order eller faktura direkt från kund-vyn skulle kunna läggas till. Detta är dock endast rekommendationer från min sida och inte från företaget. De kommer att utvärdera produkten och sedan ta ställning till vad som behöver läggas till eller ändras.

8.3 Slutsats

Att skapa ett affärssystem helt utefter användarens krav har visat sig vara både ett givande och krävande arbete. Att kunna arbeta i nära samarbete med kunden och användaren har varit en stor tillgång. Utvecklingen har varit krävande av den anledningen att användaren och jag inte alltid pratat samma språk. I detta fall har användningsfallsdiagrammen som skapats enligt UP varit till stor hjälp. Användaren kunde genom de diagrammen få en bättre bild av vad ett krav egentligen är.

Användarna hade tidigare erfarenhet av olika affärssystem och visste därigenom vad de ville ha men inte riktigt vad som krävdes för att genomföra de önskemålen. Genom att använda UP har jag kunnat bygga systemet steg för steg och hela tiden visa för och ta hjälp av användaren för att komma på rätt spår.

Hela iden bakom Software Engineering som jag beskrivit i bakgrunds kapitlet har visat sig förenkla utvecklingen mycket. Eftersom jag gick in i projektet med vetskapen om att det är svårt att både hamna på rätt och samma nivå som användaren, så kunde jag utnyttja de tekniker som idag finns dokumenterade i form av processer, modeller och metoder.

Resultatet har blivit ett lätthanterligt, utbyggbart system som företaget är mycket nöjda med.

9 Referenser

[1] Kent Beck. *Extreme Programming Explained.* Addison-Wesley, 1st edition, 1999.

[2] Barry W Boehm. *A Spiral Model of Software Development and Enhancement.* IEEE Computer, 1988.

[3] John Deacon, *Model-View-Controller Architecture.* 2005.

[4] Ramez Elmasri, Shamkant B. Navathe. *Fundamentals of Database Systems.* Addison-Wesley, 3d edition, 2000.

[5] Wilbert O. Galitz. *The Essential Guide to User Interface Design.* Wiley, 2nd edition, 2002.

[6] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software.* Addison-Wesley, 1995.

[7] Carlo Ghezzi, Mehdi Jazayeri and Dino Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, 2nd edition, 2003.

[8] Leif Ingevaldsson. *JSD - metoden för systemutveckling.* Studentlitteratur, 1985.

[9] Ivar Jacobson, Grady Booch, James Rumbaugh. *The Unified Software Development Process.* Addison-Wesley, 1999.

[10] Glenn E. Krasner, Steven T. Pope. *A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System.* Parc Place Systems, 1988.

[11] Hans Lunell. *Fyra rundor med RUP.* Studentiltteratur, 2003.

[12] Ruth Malan, Dana Bredemeyer. *Functional Requirements and Use Cases.* Bredemeyer Consulting White Paper, 2001.

[13] Elaine L. May, Barbara A. Zimmer. *The Evolutionary Development Model for Software.* The Hewlett-Packard Journal, 1996.

[14] James Rumbaugh, Ivar Jacobson, Grady Booch. *The Unified Modeling Language Reference Model.* Addison-Wesley, 1999.

[15] Walt Scacchi, *Process Models in Software Engineering.* University of California, Irvine 2001.

[16] Ian Sommerville. *Software Engineering,* Addison-Wesley, 6th edition, 2001.

[17] Karl E. Weigers. *In Search of Excellent Requirements.* The Journal of Quality Assurance Institute, January 1995.

A Användarmanual System

Användarmanual

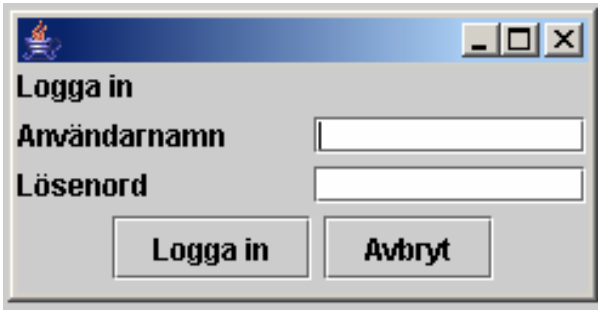
för

Ponderator AB

Affärssystem

1 Logga in

När systemet startas upp visas en ruta för inloggning (se Figur 9.1). Inga andra delar av systemet är tillgängligt innan inloggningen har slutförts.



Figur 9.1 Inloggnings ruta

För att logga in till systemet gör följande steg:

1. Skriv in användarnamn (som erhållits av system ansvarig)
2. Skriv in lösenord (som erhållits av systemansvarig)
3. Klicka på ”Logga in”

Om användarnamn och lösenord är korrekt försvinner inloggningsrutan och systemets huvudfönster visas.

Om användarnamn och/eller lösenord är felaktiga visas inloggningsrutan fortfarande med texten ”Försök igen” Upprepa då steg 1-3 ovan. (Tänk på att inloggningen är skiftläges känsligt)

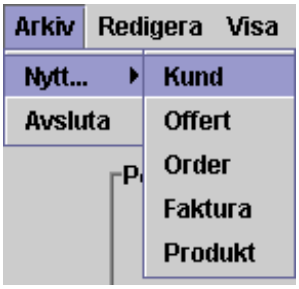
2 Menyer i huvudfönstret

Systemets huvudfönster innehåller tre menyer (se Figur 9.2). Menyerna används förr att få tillgång till de olika arbetsmomenten som stöds av systemet.

Figur 9.2 Menyer i huvudfönstret

2.1 Meny Arkiv

Meny ”Arkiv” Innehåller val för att skapa nya objekt (se Figur 9.3). Samt ett meny val för att avsluta systemet. För att skapa ett nytt objekt välj Arkiv -> Nytt -> ”Objektet som önskas”



Figur 9.3: Menyn Arkiv

2.2 Menyn Redigera

Menyn redigera i huvudfönstret (se Figur 9.4) Innehåller val för att öppna ett specifikt objekt, samt även val för att ta bort en produkt. För att visa eller redigera ett specifikt objekt välj Redigera -> ”Objektet”

Figur 9.4: Menyn Redigera

2.3 Menyn Visa

Visa menyn i huvudfönstret innehåller val för att visa listor på befintliga objekt i databasen (se Figur 9.5). För att se en lista på ett visst objekt välj Visa -> Alla ”Objektet”

Figur 9.5: Menyn Visa

3 Kund

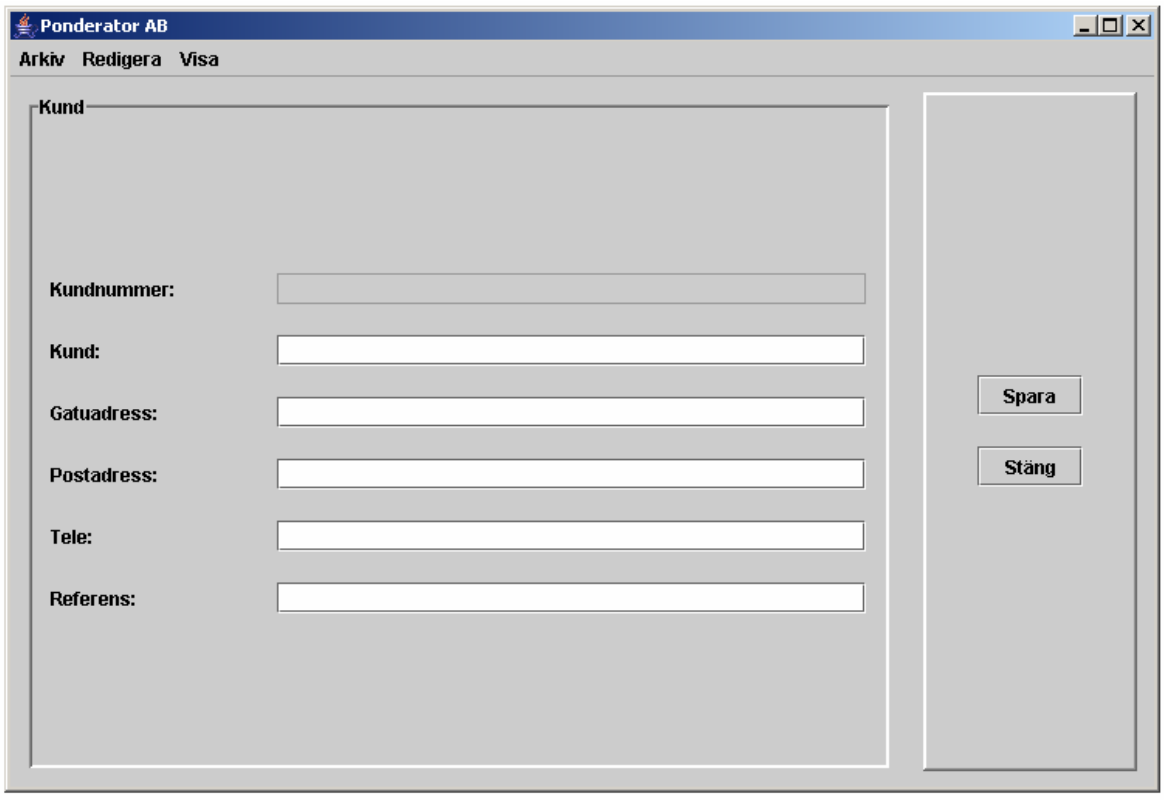
Systemet stödjer ett kundregister. För att behandla kundregistret följ kommande instruktioner.

3.1 Skapa ny kund

För att lagra kunder i registret behöver information om kunden matas in i formuläret för kunden (se Figur 9.6). Följande steg krävs för att en kund ska läggas till.

1. Välj Arkiv -> Nytt -> Kund
2. Fyll i Kundens/Företagets namn (obligatoriskt)
3. Fyll i Adress (obligatoriskt)
4. Fyll i Postadress (obligatoriskt)
5. Fyll i Telefonnummer (valfritt)
6. Fyll i Referens på företaget (valfritt)
7. Spara

Om alla obligatoriska fält har fyllts i så har kunden lagts till i registret. Om kunden redan fanns i registret (namnet) så ändra namnet och försök igen.



Figur 9.6: Kundformulär

3.2 Visa eller Redigera kund

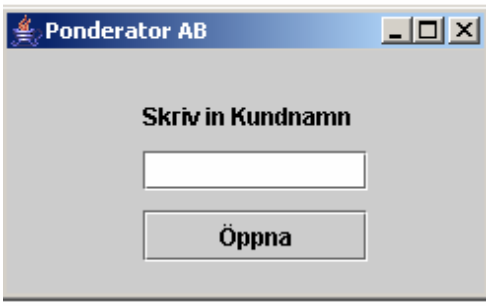
För att ändra information om kunder, följ kommande steg.

3.2.1 Alternativ 1

Välj Redigera -> Kund

I formuläret för kunden (se Figur 9.6).

1. Fyll i Kundens/Företagets namn i popupfönstret (se Figur 9.7)
2. Ändra informationen enligt önskemål
3. Spara



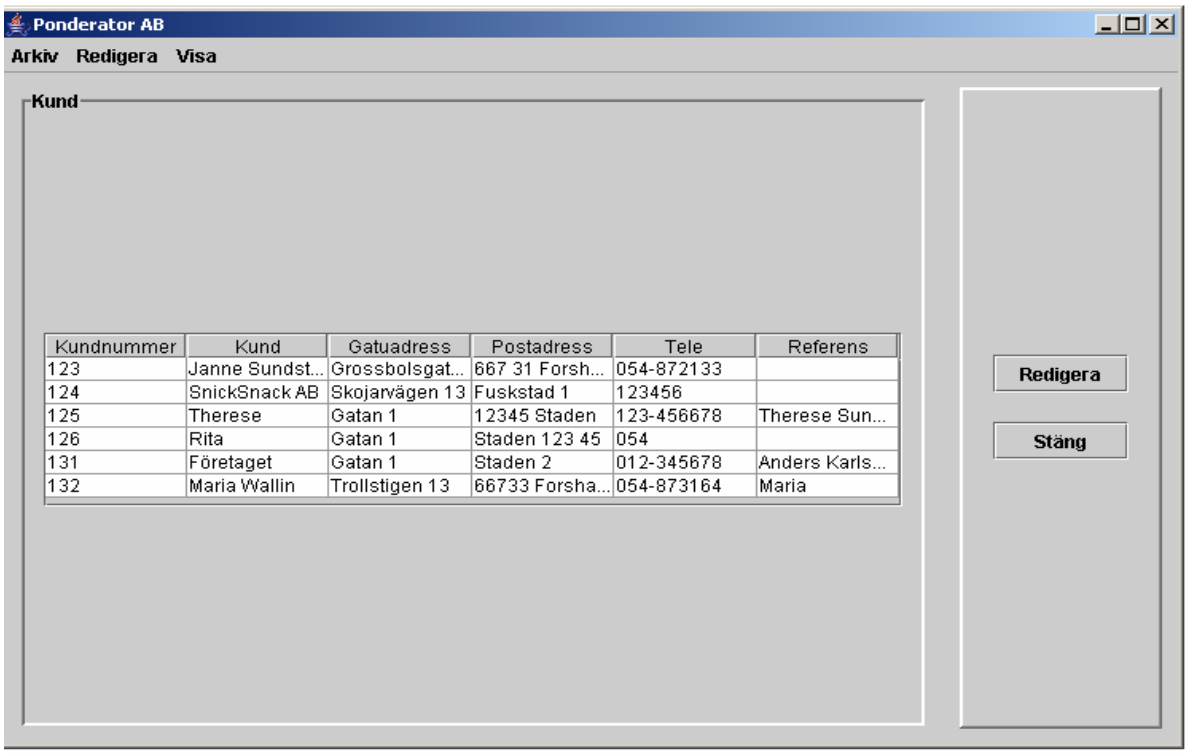
Figur 9.7: Fönster för inmatning av kund/företags namn

Om alla obligatoriska fält har fyllts i så har kunden lagts till i registret.

3.2.2 Alternativ 2

Välj Visa -> Alla Kunder

1. Markera önskad kund i listan (se Figur 9.8)
2. Tryck på ”Redigera”
3. Ändra informationen enligt önskemål
4. Spara



Figur 9.8: Lista på alla kunder

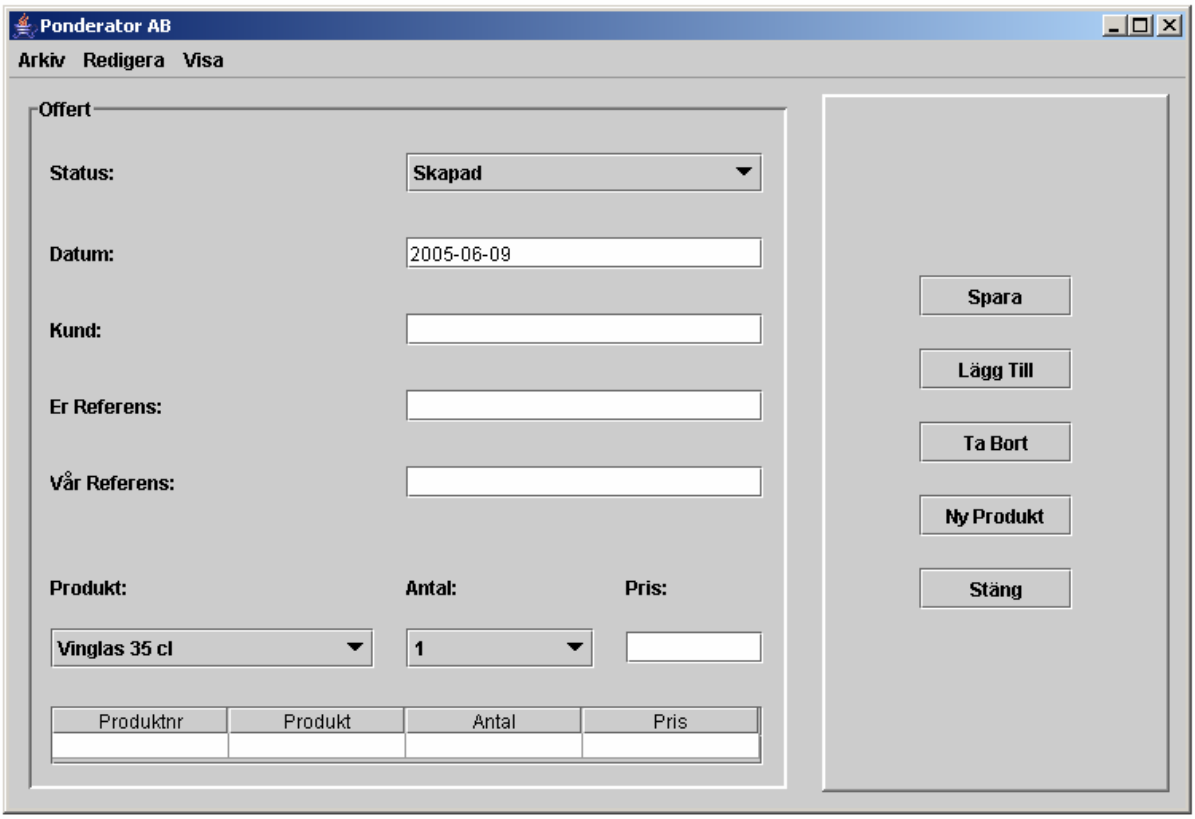
4 Offert

Systemet stödjer att skapa och redigera offerter. För att använda denna funktion följ stegen nedan.

4.1 Skapa Offert

För att lagra offerter i registret behöver information om offerten matas in i formuläret för offerten (se Figur 9.9). Följande steg krävs för att en offert ska läggas till.

1. Välj Arkiv -> Nytt -> Offert
2. Fyll i Kundens/Företagets namn (obligatoriskt)
3. Fyll i namnet på er referens (valfritt)
4. Fyll i namnet på kundens referens (valfritt)
5. Välj Produkt i listan
6. Välj Antal produkter
7. Sätt pris på produkten
8. Tryck på ”Lägg till” för att flytta produkten till listan.
9. Upprepa steg 5-8 för alla produkter
10. Spara



Figur 9.9: Offertformulär

4.2 Visa eller Redigera offert

För att ändra status på offerten följ stegen under Alternativ 1 eller Alternativ 2. (Kan också användas för att titta på offerten)

4.2.1 Alternativ 1

För att visa och/eller redigera en faktura följ kommande steg:

Välj Redigera -> Offert

1. Fyll i Offertnummer i popupfönstret (se liknande Figur 9.7)
2. Om offerten har skickats visas ett meddelande. Klicka då på ”OK”
3. Ändra Statusen på offerten
4. Tryck på ”Ändra” (eller ”Stäng” om inget ska ändras)

4.2.2 Alternativ 2

Välj Visa -> Alla Offerter

1. Markera önskad offert i listan (se Figur 9.8)
2. Tryck på ”Redigera”
3. Ändra informationen (status) enligt önskemål
4. Tryck på ”Ändra” (eller ”Stäng” om inga ändringar gjorts)

5 Order

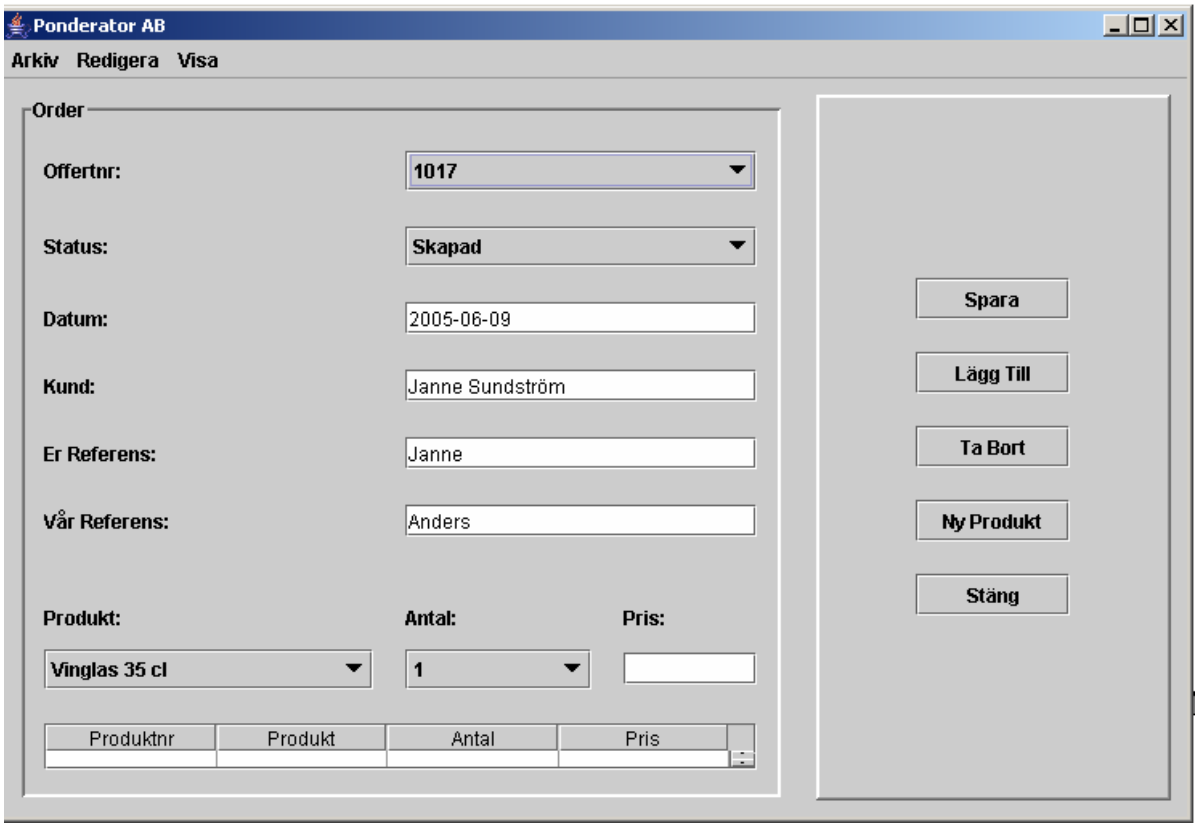
Systemet stödjer att skapa och redigera offerter. För att använda denna funktion följ stegen nedan.

5.1 Skapa Order

För att lagra order i registret behöver information om ordern matas in i formuläret för ordern (se Figur 9.10Figur 9.9). Följande steg krävs för att en order ska läggas till.

5.1.1 Alternativ 1 (Utan Offert)

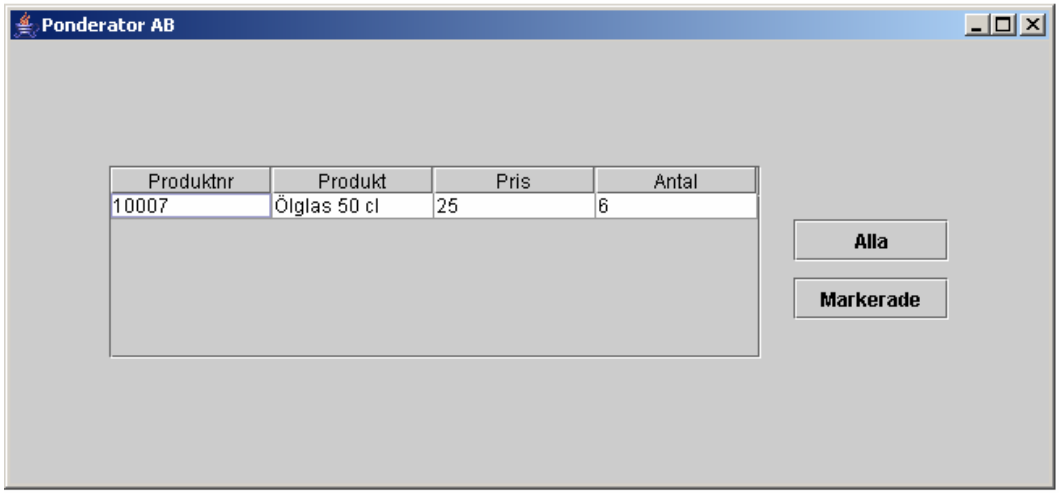
1. Välj Arkiv -> Nytt -> Order
2. Fyll i Kundens/Företagets namn (obligatoriskt)
3. Fyll i namnet på er referens (valfritt)
4. Fyll i namnet på kundens referens (valfritt)
5. Välj Produkt i listan
6. Välj Antal produkter
7. Sätt pris på produkten
8. Tryck på ”Lägg till” för att flytta produkten till listan.
9. Upprepa steg 5-8 för alla produkter
10. Spara



Figur 9.10: Orderformulär

5.1.2 Alternativ 2 (Med Offert)

1. Välj Arkiv -> Nytt -> Order
2. Välj offertnummer i listan (se Figur 9.10)
3. Välj Produkter i listan (från offerten se Figur 9.11). Välj ”Alla” för alla produkter eller markera produkter genom att klicka i listan med skifttangenten nedtryckt och klicka på ”Markerade”
4. Upprepa steg 5-8 under alternativ 1 för att lägga till fler produkter
5. Spara



Figur 9.11: Lista på produkter från offert

5.2 Visa eller Redigera order

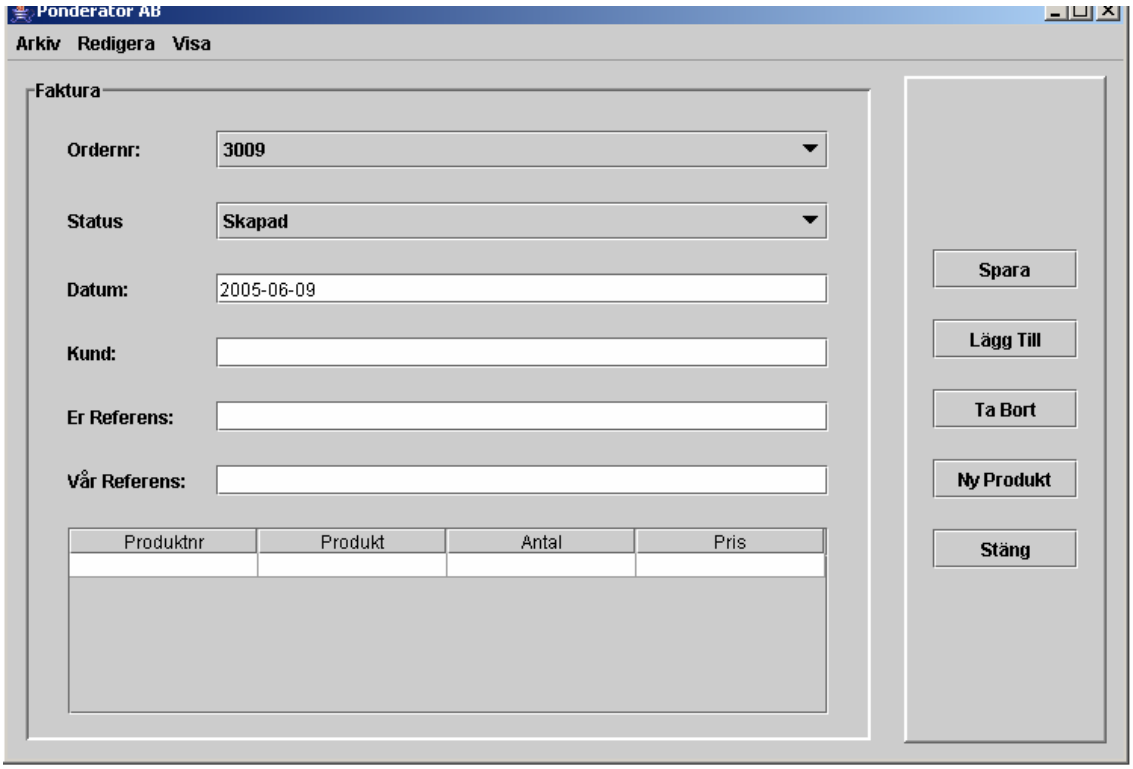
Se under ”Visa eller Redigera Offert”.

6 Faktura

Systemet stödjer även fakturering. Följ kommande steg för att skapa och hantera fakturor.

6.1 Skapa Faktura

1. Välj Arkiv -> Nytt -> Faktura
2. Välj ordernummer i listan (se Figur 9.12)
3. Välj Produkter i listan (från ordern se Figur 9.11). Välj ”Alla” för alla produkter eller markera produkter genom att klicka i listan med skifttangenten nedtryckt och klicka på ”Markerade”
4. Spara



Figur 9.12: Fakturaformulär

6.2 Visa eller Redigera faktura

Se under ”Visa eller redigera offert”.

7 Produkt

Systemet innehåller även ett produktregister. För att hantera produkter följ stegen nedan.

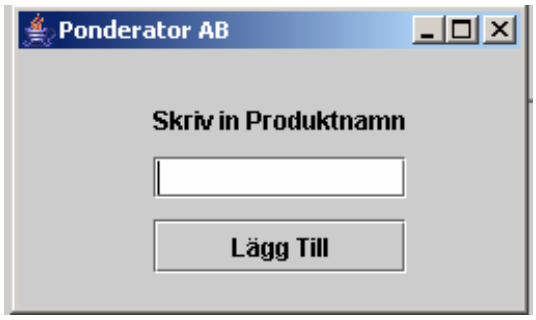
7.1 Lägg till ny produkt

Det finns två varianter på att lägga till produkter i systemet

7.1.1 Alternativ 1

Produkter kan läggas till genom följande steg:

1. Välj Arkiv -> Nytt -> Produkt
2. Skriv in produktnamn i popupfönstret (se Figur 9.13)
3. Klicka på ”Lägg Till”



Figur 9.13: Popupfönster för produkter

7.1.2 Alternativ 2

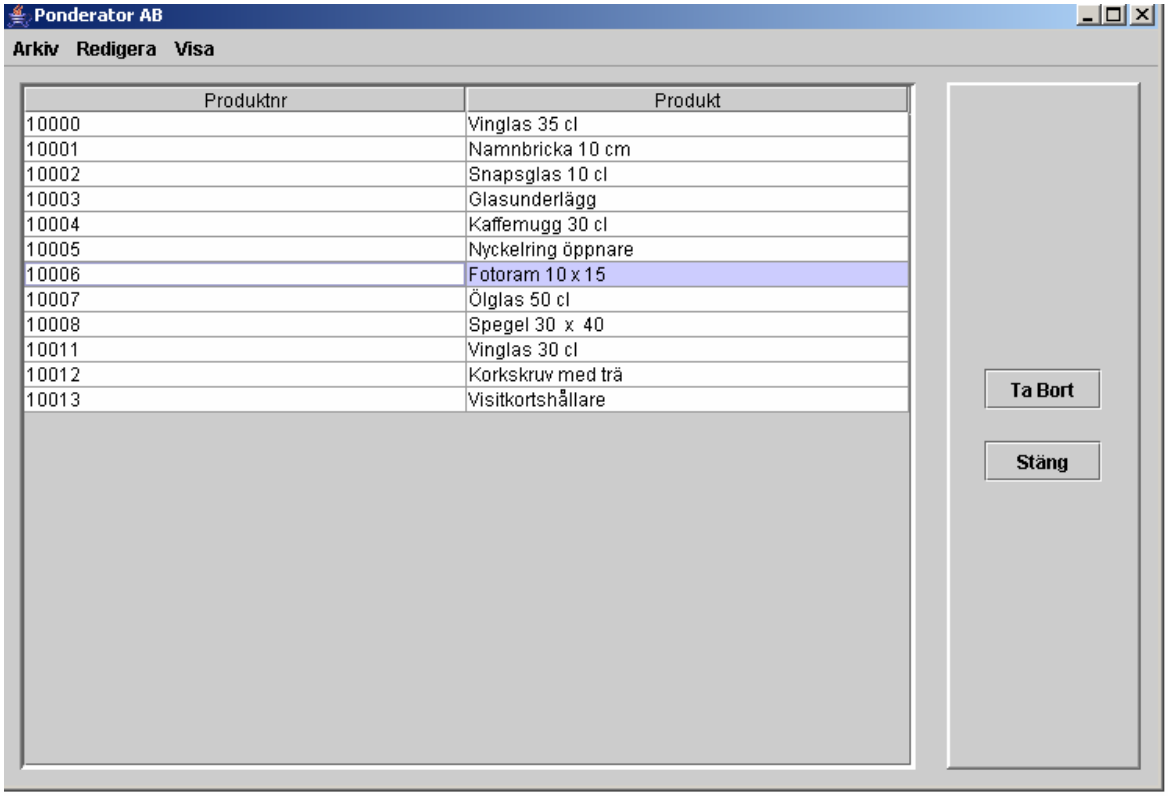
Produkter kan även läggas till genom följande steg:

4. Välj Arkiv -> Nytt -> Offert/Order
5. Klicka på ”Ny Produkt”
6. Skriv in produktnamn i popupfönstret (se Figur 9.13)
7. Klicka på ”Lägg Till”

7.2 Visa eller Ta bort produkt

Systemet kan behöva uppdateras då produkter inte används längre. Följ stegen nedan för att ta bort en produkt.

1. Välj Visa -> Alla Produkter
2. Markera Produkt (se Figur 9.14)
3. Klicka på ”Ta Bort”



Figur 9.14: Lista på alla produkter

B Användarmanual Excelmallar

Användarmanual

för

Ponderator AB

Excelmallar

1 Excelmallar

Ett antal mallar har skapats för att få en bra struktur på företagets Offerter, Order, Ordererkännande och fakturor. Följande avsnitt förklarar de steg som krävs för att använda en mall och infoga produkter direkt från databasen.

1.1 Öppna Excelmallen

För att skapa en offert, order, ordererkännande eller faktura, öppna motsvarande mall. Spara om mallen med motsvarande unikt nummer som objektet har (offerten etc.).
(Tänk på att alltid öppna mallen och inte senaste fakturan, annars fungerar inte summeringen)

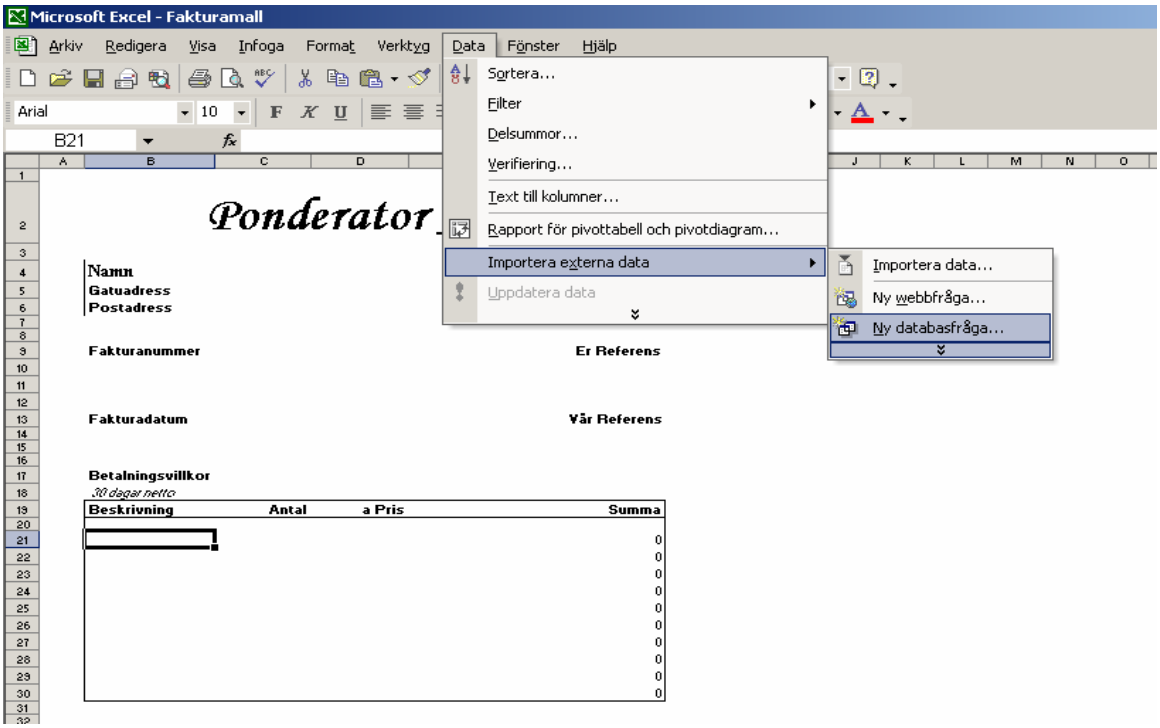
1.2 Infoga kundinformation

I mallen finns fält för att skriva in kundens information samt betalningsvillkor. Detta görs manuellt.

1.3 Hämta data från databas

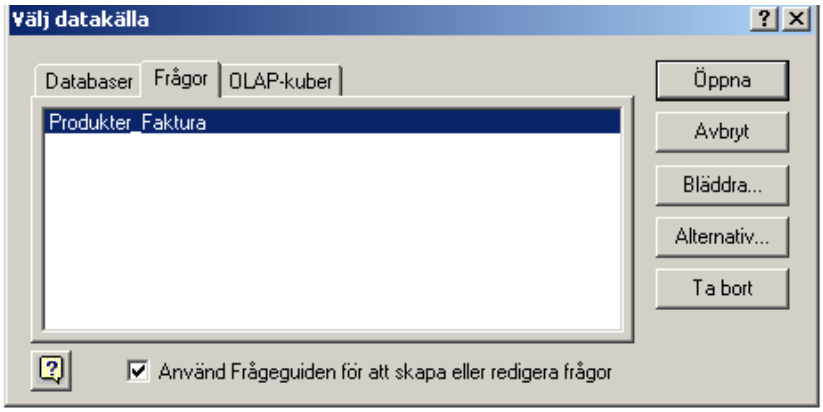
De objekt som innehåller många produkter kan med fördel hämta dessa direkt från databasen. Detta görs med hjälp av MS Query. Denna åtgärd kräver ett antal steg som följer:

1. I Excel välj Data -> Importera externa data -> Ny Databasfråga



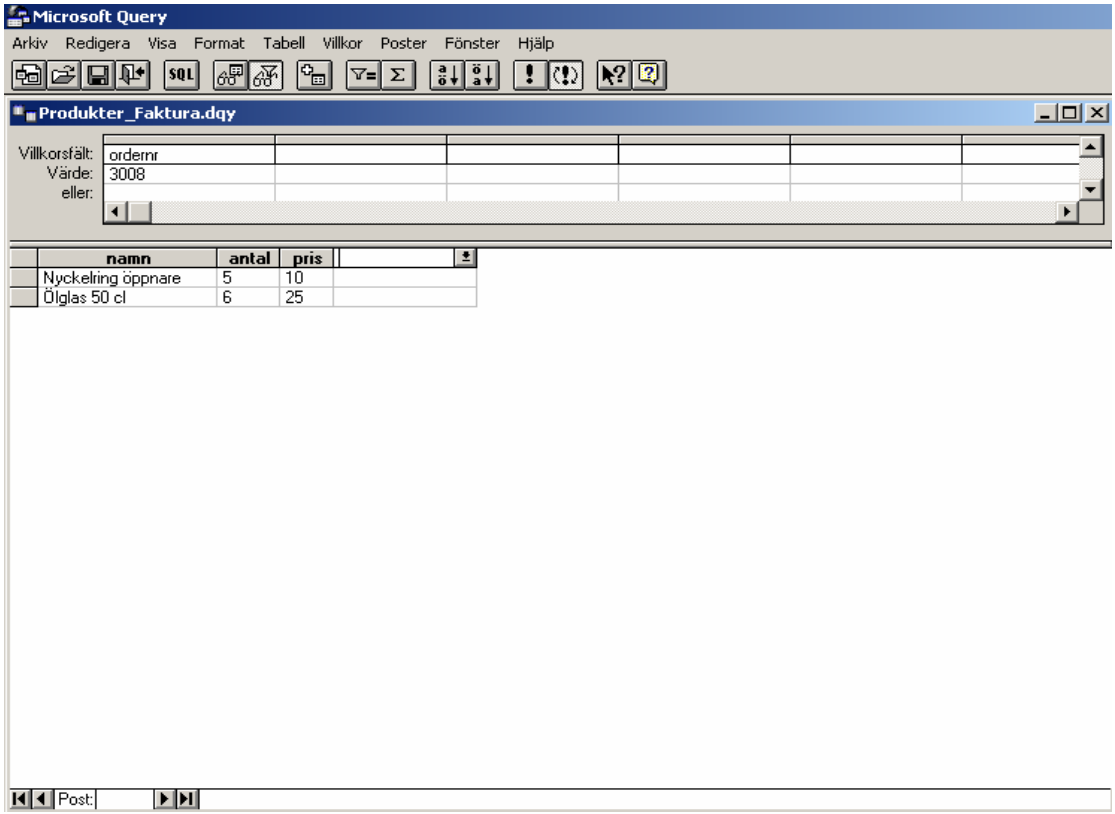
Figur 9.15: Hämta data i Excel

2. I fönstret Välj Datakälla Klicka på fliken Frågor
3. Välj den fråga som matchar dokumentet
4. Klicka på ”Öppna”



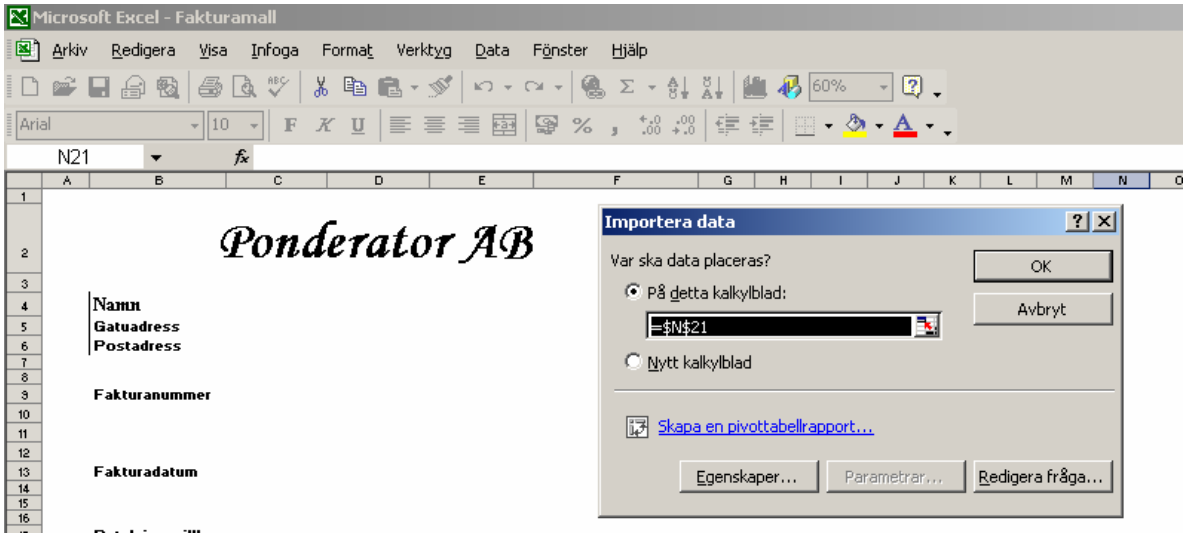
Figur 9.16: Välj databasfråga

5. MS Query öppnas, skriv in rätt unikt nummer under ”Värde”
6. Välj sedan Arkiv -> Returnera data till Microsoft Excel



Figur 9.17: MS Query

7. Nu ska data importeras, klicka på den röda pilen i ”Importera data” fönstret
8. Markera det område där produkterna ska infogas
9. Klicka på ”OK”



Figur 9.18: Välja område för data

Produkterna har nu importerats från databasen. Automatisk summering av produkterna görs.