Garantiregistreringsapplikation

Skapa garantikvitton med hjälp av befintlig kund- och produktdata

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Mutltmediateknik

Examensarbete:
Anders L. Bergh
Daniel Nord

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds Universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Media-Tryck
Biblioteksdirektionen
Lunds Universitet
Lund 2006

Sammanfattning

Vårt examensarbete gick ut på att skapa en prototyp på en applikation till Team Sportia, Hässleholm, som skulle underlätta garantiregistering vid cykelförsäljning. Den skulle göra sökning och sammställning av information från en Oracle-produktdatabas och en IBM-kunddatabas. Informationen skulle sedan skrivas ut som ett garantikvitto samtidigt som den skulle sparas i en Oracle- eller IBM-databas.

Vi skulle kort sagt skapa en applikation som underlättade garantiregistreringen för Team Sportias anställda.

Eftersom vi inte kunde testa vår applikation mot Team Sportias miljö behövde vi skapa en med hjälp av IBMs respektive Oracles gratisversioner av sina databaser. De hade tillräcklig funktionalitet för vårt testmiljösyfte. Garantikvittot skulle innehålla all den information som finns på kvittona idag. För att vara säkra på det undersöktes Team Sportias tre olika garantikvitton för att kunna skapa ett liknande i applikationen.

Vår applikation fungerar utifrån de premisser vi hade från början. Vad vi inte hade fått reda på var att den enda möjligheten att nå databaserna var att installera applikationen på en server med Citrix. Det här betyder att vi inte kunnat testa vår anslutning mot Team Sportias databaser. Men det ska inte vara något större problem om man vill få över systemet till Citrix-servern. Vi har inte testat anslutningarna på Team Sportia på grund av att systemteknikerna inte haft tid till att hjälpa oss.

Vi mätte hur mycket bättre tiden blev genom att skriva garantikvittot med vår applikation. Det tog 3,5 minut att skriva det gamla garantikvittot för hand men med vår prototyp tog det drygt 30 sekunder att skapa ett med samma information. Efter att drivrutinerna lästs in tog det knappt 20 sekunder att skapa kvittot.

Vi uppnådde vårt syfte att skapa ett hjälpmedel som underlättade för Team Sportias personal, men vi nådde inte riktigt vårt mål att skapa ett prototypprogram som fungerade i deras befintliga Windowsmiljö.

Nyckelord: Garantiregistreringssystem, Visual C++, IBM DB2, Oracle 10g

Abstract

Our degree project was to create a prototype application for Team Sportia, Hässleholm. The prototype should facilitate guarantee registration in bike sales. The application should use information from existing Oracle and IBM databases to print a warranty receipt and save the information in either an Oracle or an IBM database.

Our problem was how to create an application that simplyfied the warranty process for the employees.

To avoid affect Team Sportia we first had to create a test environment with the help of free versions of both Oracle's and IBM's databases. They had enough function for our purpose. The guarantee receipt was supposed to use all information from the receipt of today. To be sure, we examined Team Sportias three receipts to make a similar one in our application.

Our application works from the premises we had from the beginning. For the communication our application did not work on Team Sportia's system due to faulty information. We were not told that the only ways to reach the databases were to install the application on a Citrix server. This means that we had not tested our application against Team Sportias databases. With the information we had gathered we concluded that this would bring no problem if you want to transfer the application to a Citrix server.

We counted how much better the registration time was with our application. Our result was satisfactory. To write the warranty receipt by hand took 3,5 minutes. Our prototype cut it down to amply 30 seconds. After the drivers were loaded it took less than 20 seconds to create and print the warranty receipt.

We achieved our object to create a tool for helping Team Sportias employees but we did not reach our goal to create a prototype program that worked in Team Sportia's existing Windows environment.

Förord

Detta exjobb baseras på en flaskhals vid försäljningen av cyklar på Team Sportia Hässleholm. Bakgrunden till exjobbet ligger i att en av exjobbsdeltagarna jobbade som cykelmontör hos Team Sportia och därigenom uppmärksammat flaskhalsen med initiativ av butiksägaren.
Problemet är att garantikvitton skrivs för hand vilket är tidsödande samtidigt som de ibland kan vara svårlästa.
Det var först nu i år som Team Sportia såg möjligheten att få en lösning som betydde snabbare garantiskrivning av cyklar. Åke Bergh som är nuvarande butikschef tog kontakt med oss då han visste att vi stod inför val av exjobb för vårterminen 2006.

Vi vill tacka Åke Bergh för att vi fick möjligheten att utveckla applikationen till Team Sportia. Tack Mats Rüder och Peter Sandegård för all hjälp med rapportskrivandet under examenstiden.

Exjobbet har lärt oss vikten av att man måste ha en bra metod och ett klart mål för att uppnå den kvalité och funktionalitet som krävs för ett bra program.

Helsingborg, Maj 2006

Anders L Bergh
Daniel Nord

Innehållsförteckning

1 Inledning

1.1 Bakgrund

I slutet av 2005 visade sportkedjan Team Sportia, Hässleholm, sitt intresse för att ge oss ett användbart och lärorikt examensarbete. Examensarbetet baserades på cykelförsäljningen inom koncernen och registrering av cykelköp.

Med hjälp av information i två databaser skulle vi skapa ett garantikvitto till kunden och samtidigt spara det i en databas.

Applikationen skulle vara Microsoft Windows-baserad och den skulle inte inverka på det befintliga systemet. Vid försäljning skulle försäljaren fylla i kundens personnummer samt cykelns EAN-nr med hjälp av en streckkodsläsare. De två sökorden hämtade resterande information i två separata databaser. Utifrån den hämtade informationen skulle den sparas i en ny databas och ett garantikvitto skulle skrivas ut. Det skulle vara lätt att anpassa applikationen till Team Sportias andra butiker eftersom Hässleholm är en av flera butiker med cykelförsäljning.

Kundregistret är den nystartade Team Sportia-klubbens register och artikelregistret är gemensamt för koncernen och används för att hitta artiklar och information.

1.2 Målsättning

Vår målsättning är att skapa ett prototypprogram som fungerar i Team Sportias befintliga Windowsmiljö som kan spara och skriva ut garantikvitton med hjälp av Team Sportias befintliga databaser.

1.3 Syfte

Syftet är att skapa ett hjälpmedel för att underlätta garantiförfarandet vid försäljning av cyklar. Huvudtanken är att applikationen ska förkorta tiden för cykelförsäljning och därmed underlätta arbetet för försäljaren.

1.4 Problemformulering

Idag skrivs Team Sportias cykelgarantikvitton manuellt, för hand, vilket tar onödig tid i anspråk och även ökar risken för misstag. Säljaren måste mata in ett flertal fält, för garantiregistrering, som redan skrivits in när kunden skaffade ett klubbkort och när cykeln registrerades i artikelregistret. Informationen ligger redan i en Oracle- och en IBM-databas som används för artikelregister respektive kundklubb.

Databaserna får inte ändras då andra applikationer är beroende av dagens upplägg.

Databaserna är placerade på en server i Mölnlycke vilket medför att anslutningen till databaserna måste ske över internet då servern är på en helt annan plats än Team Sportia, Hässleholm. Eftersom de ligger i en kritisk miljö får vi inte testa att hämta data ifrån dem.

Applikationen ska utföra två saker. Det är att skriva ut ett garantikvitto och att spara garantiinformationen i en databas. Garantikvittot ska ges till kunden och informationen som sparas i databasen är butikens exemplar av kvittot.

Utifrån denna information har vi kommit fram till följande problemformulering: Att skapa en applikation som hjälper personalen på Team Sportia att generera garantikvitton på kortare tid och samtidigt bygger bort felkällor.

För att lösa vårt problem måste följande delar lösas:
- Skapa en testmiljö som liknar Team Sportias för att kunna testa databasanslutningar.
- Sammanställa och presentera information från en Oracle 10g- och en IBM DB2-databas utifrån artikel- och kundnummer hämtad från streckkodsläsare.
- Spara garantiinformation i en Oracle 10g- eller en IBM DB2-databas.
- Skapa en garantiutskrift med den information Team Sportia har idag.

1.5 Avgränsningar

Vi kommer inte att utreda vilken typ av applikation, t.ex. webbaserad, då detta kravet var en klientbaserad Windows-applikation.

Prototypen kommer inte att kunna hantera felaktigt inskrivna EAN-nr eller kund-id från databasen. Om en cykel fått ett felaktigt EAN-nr kan antingen information om fel cykel hämtas eller hittas ingen information alls. I fallet att ingen information finns visas ett felmeddelande.

Ett krav är att lösningen ska vara en applikation för Microsoft Windows som inte ändrar eller påverkar nuvarande program.

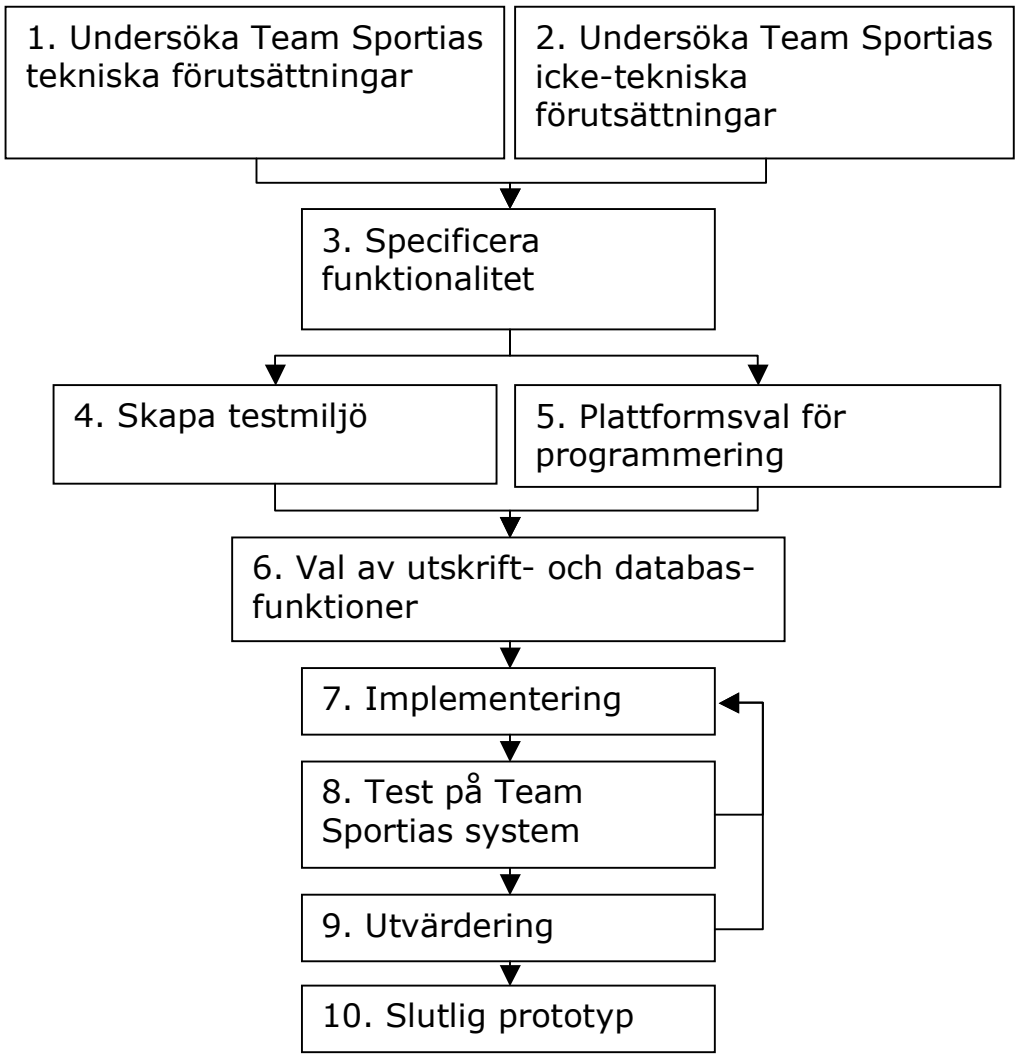
Vi kommer inte att skapa ett gränssnitt som är anpassat till Team Sportias grafiska profil eftersom detta arbete hade gått ut över vår möjlighet att arbeta med funktionaliteten.

2 Metodik

Vi kommer att börja med att undersöka Team Sportias förutsättningar såväl tekniska som icketekniska.

Därefter kommer vi att ta fram en testmiljö där vi kan utveckla vår prototyp samt välja vilken programmeringsmiljö och språk vi ska använda. Vi behöver även välja på vilket sätt vi ska hämta informationen från databaserna och hur vi ska skriva ut till skrivaren. När väl detta är gjort kan vi påbörja programmering och implementering av systemet. I slutskedet kommer vi testa prototypen på Team Sportias system för att därmed upptäcka eventuella missar i kodning och interaktion.

Slutligen utförs en utvärdering och eventuella justeringar av systemet. Om vårt arbete går enligt planen kommer det leda till en slutgiltig prototyp för ett garantikvittosystem. Figuren nedan (figur 1) visar hur vår metodik ser ut.



Figur 1

2.1 Undersöka Team Sportias Tekniska förutsättningar

Vi behöver veta Team Sportias tekniska förutsättningar såsom operativsystem, hårdvara, information om kunddatabas- och produktdatabaserna. Eftersom Team Sportia använder streckkodsläsare blir även detta en förutsättning, d.v.s. ta reda på hur den skickar texten och vad som händer när textsträngen är slut. Databas och operativsystemsinformationen kommer vi använda vid skapandet av en testmiljö för att kunna avgöra om man behöver installera andra drivrutiner än de som är installerade som standard. Streckkodsläsartestet utförs så att vi kan programmera gränssnittet funktionellt för användaren.

2.1.1 Streckkodsläsarens funktion

För att ta reda på hur man läser in informationen med en steckkodsläsare måste vi testa hur en sådan fungerar. Vi vill veta hur informationen som läses in skickas till datorn. Måste man installera drivrutiner och omvandla informationen från streckkodsläsaren från t.ex. ett binärt format till ett format man kan använda för att söka efter information i databasen? För att förstå detta bättre kommer testen genomföras med hjälp av butiksägaren. Man hade kunnat läsa manualer men vi kommer att testa streckkodsläsaren i praktiken för att få en bättre förståelse.

2.1.2 Ta reda på operativsystem och hårdvara

Vi behöver ta reda på vilket operativsystem som används. Det här åstadkoms genom att be butiksägaren visa en dator där applikationen kommer att köras när den är utvecklad. Vi kommer där att analysera vilket operativsystem som används och vilken hårdvara som körs på den. Genom att analysera själva istället för att be butiksägaren att kontrollera, kan vi själva dra slutsatser av undersökningen. Vi kan få de uppgifter om datorn som vi vill ha utan att få eventuella felaktigheter, som fallet varit om butiksägaren inte varit insatt i datorer.

Vi måste även få information om var servern med databaserna finns någonstans och hur vi skall ansluta till dem. Det här är ingenting vi själva kan kolla upp utan är information vi måste få från de IT-ansvariga.

För att veta prestanda på Team Sportias datorer kommer vi att fråga butiksägaren om den hårdvara som finns installerad på datorerna. Det som är mest intressant är processor och ramminne eftersom det är de delarna som har mest betydelse om applikationen kommer att fungera.

2.1.3 Tabellernas struktur i databaserna

För att skapa en korrekt testmiljö blir vi tvungna att ha rätt tabellnamn, kolumnnamn och datatyper. Detta kommer att tas fram genom att kontakta de ansvariga för databaserna.

2.2 Undersöka Team Sportias icke-tekniska förutsättningar

I detta moment kommer vi att undersöka hur det praktiskt går till för att genomföra en garantiregistrering av en cykel. Det vi kommer att undersöka är genomförande av registrering samt tiden det tar för att registrera.

2.2.1 Befintlig registreringstid

För att veta befintlig registreringstid kommer vi använda oss av ett tidtagningsur och ett butiksbiträde som skriver ut ett garantikvitto. Vi väljer att genomföra detta test på detta vis för att få en verklig uppfattning om tiden det tar för att skriva ett garantikvitto för hand. Eftersom vi väljer ett butiksbiträde som dagligen skriver garantikvitton behöver vi inte testa på fler än en person.

2.2.2 Garantikvittots utseende

För att veta hur garantikvittot vi ska skapa ska se ut måste vi kolla på dagens garantikvitton och även föra en dialog med Team Sportia. En skiss på kvittot kommer att tas fram för att ha något att ha som mall vid utvecklandet av koden senare. De krav vi har på garantikvittot är att det ska innehålla samma information som dagens kvitto.

2.3 Specifik funktionalitet

En kravspecifikation skapas för att ha ett dokument som konkretiserar det som applikationen ska klara av. Vi får dessutom en övergripande bild av det vi ska skapa. Skapandet av kravspecifikationen fastställer allting så att inget extra ska komma in senare under processen från Team Sportia eller oss själva.

Kravspecifikationen kommer att skapas genom möten med Team Sportia. Vårt mål är inte en komplett kravspecifikation utan en som fastställer grundläggande

utseende på applikationen samt de databasfält som ska användas. Även utseendet på garantikvittot kommer att vara med.

2.4 Skapa testmiljö

Vi behöver skapa en testmiljö där utvecklingen kan ske utan att störa driften i Team Sportias befintliga system. I miljön kommer vi att installera två gratisversioner av databasservrarna. Det är tänkt att testmiljön ska likna Team Sportias plattform, målplattformen, för att vara säker på att komplikationerna blir så få som möjligt vid övergången till det riktiga systemet. Applikationen, som kommer att ligga på en separat dator, kommer att ansluta till databaserna, som ligger på en annan av våra datorer, över ett internt nätverk. Det kommer att vara den största skillnaden jämfört med när vi senare ska testa mot Team Sportias riktiga server.

2.4.1 Installera testdatabaser

För att utvecklingen inte ska störa Team Sportia onödigt mycket när man testar anslutningen till databaserna kommer två testdatabaser skapas – en Oracledatabas 10g och en IBM DB2-databas. Eftersom de kommersiella versionerna är väldigt dyra kommer de gratis Express-versionerna av respektive databasmotor att användas. Installationsfiler till databaserna finns på Internet för installation.

2.4.2 Skapa databaser och deras tabeller i testdatabaserna

Utifrån den information vi fått tidigare från Team Sportia skapar vi databaser med deras tabeller. Det här gör vi med de grafiska gränssnitt som IBM och Oracle tillhandahåller i sina installationspaket.

2.5 Plattformsval för programmering

Val av plattform för programmering innefattar två delar; val av utvecklingsmiljö samt val av programspråk. Alla utvecklingsmiljöer klarar inte alla språk och ett visst språk fungerar kanske bara i en utvecklingsmiljö. Valet av både programspråk och utvecklingsmiljö kommer att tas samtidigt genom diskussion mellan oss två studenter och granskning av de språk vi kommer fram till att vi har möjlighet att använda. Vi strävar efter att ha så få utvecklingsmiljöer som möjligt för att underlätta utvecklingen och eventuella kostnader för Team Sportia.

2.5.1 Val av utvecklingsmiljö

Utvecklingsmiljön behöver inte nödvändigtvis vara enkel att arbeta i. Dock är stor funktionalitet samt bra dokumentation nödvändigt då vi kommer att utveckla funktioner som vi inte använt innan. Det är då en stor fördel om bra dokumentation finns för att leda oss genom utvecklingen. Priset för utvecklingsmiljön är en faktor vi tittar på för den händelse att Team Sportia skulle köpa in en licens i ett senare skede.

2.5.2 Val av programspråk

Det viktigaste var att vi skulle ha tidigare kunskaper i programspråket. Förutom det ställde projektet följande krav på språket:

- Objektorienterat
- Bra möjligheter att enkelt skapa databaskopplingar
- Skriva ut utan att få upp en dialogruta
- Bra dokumentation
- Enkla metoder att bygga upp grafiskt gränssnitt och interaktivitet.

2.6 Val av utskrifts- och databasfunktioner

När valet av programspråk och utvecklingsmiljö är gjort kommer vi att välja utskrifts- och databasfunktioner för att kunna skriva ut respektive hämta information från databaserna. Det här är de avancerade delarna i applikationen och behöver undersökas för att ha en bra bild av hur det ska fungera. Vi kommer att utnyttja kodexempel från litteratur samt forum på internet. Vidare behöver vi veta hur tekniken fungerar som extra hjälp under utvecklandet.

2.6.1 Utskriftsbibliotek

Vi vill hitta en lämplig utskriftsfunktion. Då vi vill ha ett objektorienterat programspråk kommer vi att leta efter ett bibliotek som innehåller funktioner för att skriva till en skrivare. Det ska gå att skriva ut en bild, placerad på en specifik plats i dokumentet, samt formaterad text. Vi vill även ha möjlighet till att skapa en ruta som ramar in kvittot. Det är de krav vi ställer för att det ska gå att skapa ett kvitto som är lättläst. Vi kommer att gå igenom litteratur samt vägledningar på internet för att hitta det bibliotek som passar oss bäst.

2.6.2 Databasfunktioner

För att komma åt data från de olika databaserna behöver vi hitta en standard som är lätt att tillämpa och modifiera efter behov. Vi kommer inte att lägga ner något arbete på att studera olika tekniker för att nå databaserna utan väljer en som klarar av att hämta och skriva till databaserna. Det enda krav vi har är att tekniken vi väljer ska fungera på både IBM- och Oracledatabasernarna.

2.7 Tillvägagångssätt för implementering

Med hjälp av det valda av programspråket och de funktioner som behövs för att få en fungerande applikation strukturerar vi upp de klasser applikationen ska bestå av. Databaserna och programfunktioner ska implementeras. Ett visuellt gränssnitt kommer att utvecklas och programmeras fram då det kommer att underlätta för de anställda när de ska interagera med programmet.

2.7.1 Skapa klasser

Vi kommer att dela upp applikationen i olika klasser för varje del som den ska utföra. Utifrån val av programspråk kan uppdelningen av klasser se olika ut. Det vi vill åstadkomma är en bra struktur och att det är lätt att utveckla olika delar i applikationen parallellt. Vi ska skapa ett flödesschema där alla tänkta klasser är med och även med ett relationsnät mellan dem. Detta kommer vi att göra för att de som ska sätta sig in i programmet skall få en god överblick över dess struktur. Det kommer även att bli lättare för oss att relatera till de olika delarna om klasserna redan är uppdelade innan själva kodandet startas.

2.7.2 Skapa ett grafiskt gränssnitt

Genom att skapa ett visuellt gränssnitt underlättas kommunikationen mellan användare och underliggande funktioner. Gränssnittet kommer att byggas för att det ska vara enkelt att förstå och hitta i vilket med största sannolikhet kommer att minska garantiregistreringstiden en del. Vi kommer att skapa gränssnittet i antingen ett program där man skapar det visuellt eller direkt i koden ifall att möjligheten saknas. Vi vill skapa det visuellt eftersom det kommer bespara oss en hel del tid i att sätta oss in i hur man placerar knappar och textfält.
Genom att studera ett företags grafiska profil kan man få fram färgval, eventuella designer som går att återanvända så som avdelningslinjers tjocklek eller ramar runt rutor och dylik. Man finner profilen på hemsidan, annonser eller i postutskick. Det gränssnitt vi ska skapa måste följa några kriterier som

underlättar förståelsen och uppfattning om gränssnittet t.ex. gruppering av funktioner.

2.7.3 Hämta information från databaserna

I programmet ska vi hämta kund- och produktdata från två olika databaser för att sedan spara den sammanställda informationen i en ny tabell i en av de två databaserna. Detta för att Team Sportia vid senare tillfällen skall kunna gå in och se alla garantikvitton som genomförts. Anslutningen kommer att ske med den metod vi tidigare kommit fram till att använda.

2.7.4 Skapa utskrift

Ett garantikvitto ska skrivas ut så att kunden får sitt exemplar. Det här är ett krav från Team Sportias sida. Kvittot ska innehålla alla de fält som dagens kvitto innehåller. Det ska även vara lätt att ändra storlek på typsnitt och radmellanrum, på kvittot, vid ett senare tillfälle. Team Sportias logotyp ska finnas med. Utskriften kommer att skapas av den valda utskriftsklassen.

2.8 Test på målsystem

Den här delen innebär att flytta applikationen från testmiljön till Team Sportias system. Målsystemet är alltså det system som applikationen ska köras på när applikationen är färdigutvecklad. Då vi vid detta stadium har information om systemen som används på deras datorer, kan vi ta reda på vad som behöver installeras för att programmet ska starta. Databasanslutningarna kommer att testas genom att mata in person- och EAN-nummer som existerar i databaserna.

2.8.1 Testa om applikationen startar

För att överhuvudtaget komma igång med testandet måste det gå att starta applikationen. Det som skulle kunna ställa till problem i den här fasen skulle kunna vara service pack eller liknande som saknades på målsystemet. Skulle det vara att applikationen inte startar vid testet innebär detta att vi går tillbaka till implementationsstadiet för att åtgärda felet. För att underlätta vid test på målsystem kommer vi göra en ren installation på en testdator för att se vad som behövs installeras utöver Windows XP för att applikationen ska starta.

2.8.2 Testa att hämta kundinformation

För att kunna se om anslutningen till kunddatabasen fungerar skriver vi in personnummer. Om kundinformationen syns i programmet är testet lyckat. Skulle ingen information visas kan det bero på att personnumret är fel, personnumret inte är inlagt eller att anslutningen inte fungerar. Om ett fel uppstår kommer man uppmärksammas på det genom ett felmeddelande. På det viset kan vi lokalisera var felet ligger.

2.8.3 Testa att hämta produktnummer

För att testa anslutningen till produktdatabasen skriver vi in ett korrekt produktnummer. Om produktinformationen hämtas är testet lyckat. Skulle det av händelse inte visas någon produktinformation kan det bero på att anslutningen inte fungerar eller fel inmatat nummer. Genom att tolka felmeddelandet kan vi komma fram till var felet ligger.

2.8.4 Test av utskrift

Vi vill se om utskriftsklassen vi skapat fungerar. Det kommer vi göra genom att skriva ut den information som hämtats när vi anslöt till databaserna. Skulle inte föregående moment fungera kommer vi att skapa en version av vårt program som kan hämta in dummytext till våra fält istället. På det här sättet kan vi ändå testa utskriftsklassens funktion utan att vara beroende av databashämtningen.

2.9 Utvärdering

Under utvärderingen kommer vi ha möte med Team Sportia för att se om det är några funktioner som behövs kompletteras, textrutor som ska lägga in eller dyl. Vi kommer även att testa gränssnittsförståelsen hos Team Sportia genom att låta dem använda applikationen. Vi kommer att utvärdera testningen då vi strävar efter en så lyckad prototyp som möjligt. Under testningen kommer vi även att be ett antal butiksbiträden testa programmet för att fastslå vad vi gjort rätt och vad som behöver justeras.

2.9.1 Test av gränssnittsförståelse

Då det är väldigt lätt att ändra om knapparnas placering samt resten av utseende går det lätt att testa olika upplägg av designen. Det ska vara ett lättförståeligt system som inte ska behöva någon direkt inlärning. För att vara säkra på att gränssnittet vi skapar fungerar som vi tänkt kommer vi att testa förståelsen av det genom ett fåtal testpersoner. Vi kommer att testa genom att ha ett dummyprogram där layouten är densamma men databaskopplingarnakommer att ersättas av färdiginmatad data. Vi kommer att användartesta dummyprogrammet på minst 3-4 butiksbiträden.

3 Genomförande

Genomförandet följer metodiken till stor del. Det som ställde till det för oss var att vi inte kunde testa våra databasanslutningar på Team Sportia. Vi hade då varit tvungna att flytta över programmet till Citrix vilket inte framkom i början av examensarbetet.

3.1 Undersöka Team Sportias Tekniska förutsättningar

Under de tekniska förutsättningarna försökte vi ta reda på den information vi behövde för att kunna skapa en applikation som underlättade vid registreringen. Av de tre punkter som finns är undersökningen av Team Sportias system den största. Den punkten hjälpte oss att skapa en likvärdig testmiljö i ett senare skede.

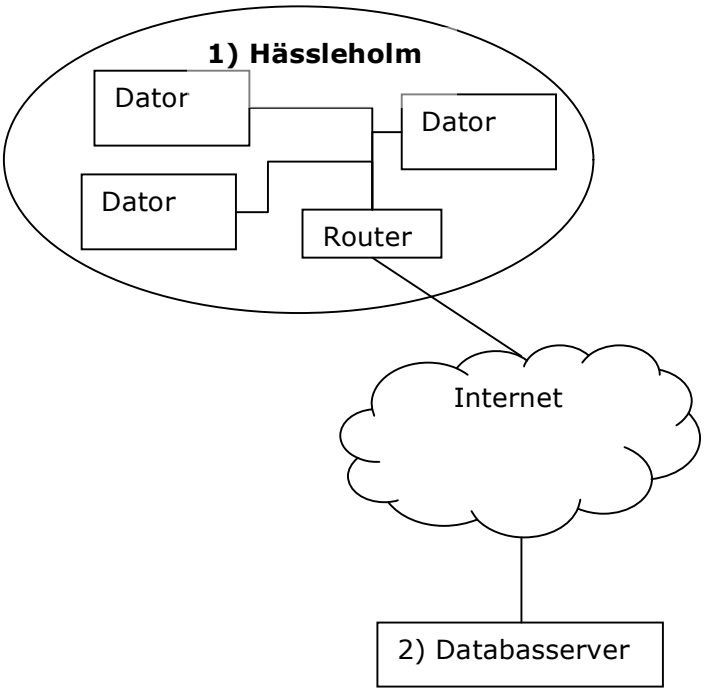
3.1.1 Streckkodsläsarens funktion

För att testa funktionen på streckkodsläsaren startade vi Notepad. Vid scannandet av ett EAN-nummer skrevs numret in i Notepad precis som om man hade skrivit siffror på tangentbordet. Vi testade även att scanna ett kundkort för att vara säker på att personnumret skrevs in på rätt sätt.
Eftersom numren skrevs in utan några extra tecken eller i något konstigt format behövdes ingen konvertering efter inläsning. Läsaren fungerar som ett tangentbord vilket betyder att vi inte behöver initiera någon enhet i programmet.

3.1.2 Ta reda på operativsystem och hårdvara

Team Sportia kör en blandmiljö med både Windows XP och Windows 2000. Datorerna är kopplade i ett nätverk och kan kommunicera via internet. När det gäller hårdvaran är alla datorer i dataparken av modernt snitt och ingen är äldre än 2 år gamla. Det här betyder att applikationen inte behöver anpassas till äldre datorer. Windows XP och 2000 är väldigt snarlika vilket kommer att underlätta vid utvecklandet.

Databaserna finns på en server i Mölnlycke. Figur 2 visar hur det är tänkt att applikationen ska fungera. I butiken i Hässleholm har Team Sportia ett nätverk som är kopplat till internet (1). Applikationen ska ligga på datorerna i Hässleholm. Databasservern (2) går att nå via internet med hjälp av dess IP-nummer. Det ska gå att komma åt databaserna från applikationen över internet om anslutningen man skapar innehåller adressen till servern.



Figur 2

3.1.3 Tabellernas struktur i databaserna

Informationen gick inte att ta reda på eftersom de IT-ansvariga inte hade tid att hjälpa oss. Därför blev vi tvungna att diskutera fram den datatyp som passade bäst för respektive kolumn. Vi kom fram till att det skulle vara lätt att ändra i den kommande koden ifall att någonting skulle vara annorlunda i de riktiga databaserna.

Kundtabellen (IBM):

Personnummer	char (10 tecken)
Förnamn	char (20 tecken)
Efternamn	char (20 tecken)
Telefonnummer	char (15 tecken)
Adress	char (30 tecken)
Postnr	int
Stad	char (15 tecken)

Artikeltabellen (Oracle):

Artikelnummer	int
Brand	char (15 tecken)
Artikelnamn	char (30 tecken)
Leverantör	char (15 tecken)
EAN-kod	char (30 tecken)
Pris	int

Garantitabellen (Oracle eller IBM)

Personnummer	char (10 tecken)
Förnamn	char (20 tecken)
Efternamn	char (20 tecken)
Telefonnummer	char (15 tecken)
Adress	char (30 tecken)
Postnr	int
Stad	char (15 tecken)
Artikelnummer	int
Brand	char (15 tecken)
Artikelnamn	char (30 tecken)
Leverantör	char (15 tecken)
EAN-kod	char (30 tecken)
Pris	int
Kampanjpris	int
Pris på tillbehör	int
Butiksnummer	char (10 tecken)
Återförsäljarnummer	char (15 tecken)

3.1.4 Sammanfattning av tekniska förutsättningar

Streckkodsläsaren fungerade betydligt lättare än vad vi förmodat. Det betyder att implementeringen inte blir svårare för att vi ska hämta in data från den.

Vi kan inte komma åt Team Sportias databaser om inte programmet läggs på Citrixservern. Vi kan inte själva lägga programmet på Citrixservern eftersom databasernas namn och uppbyggnad var svårare att ta reda på än väntat.

3.2 Undersöka Team Sportias icke-tekniska förutsättningar

Vi ville ha reda på tiden det tog att registrera dagens kvitto, samt hur tillvägagångssättet gick till för att ha någonting att jämföra mot efter att applikationen implementerats.

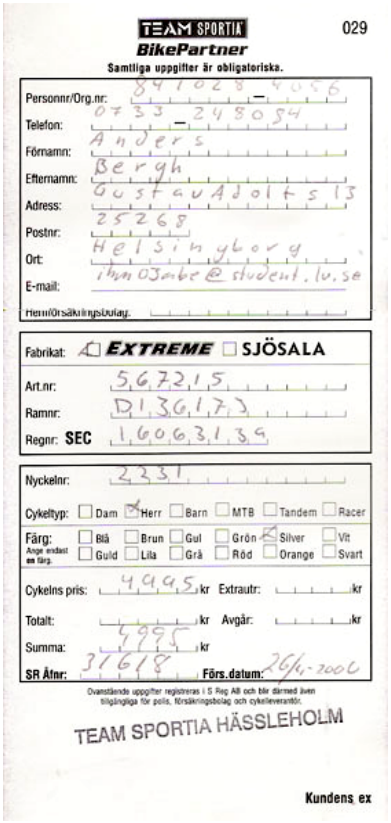
3.2.1 Befintlig registreringstid

Team Sportia skriver antingen sina garantikvitton på Excel eller för hand. Huvuddelen skrivs för hand. Det finns olika garantikvitton beroende på vilket

märke det är på cykeln. Antalet fält skiljer sig inte och inte heller vilken information som skrivs ner. Det är i princip bara layouten samt garantitexterna som skiljer sig. Det kvitto vi kommer att studera gäller för cyklar av märket Extrem och Sjösala och kan ses längst ner på denna sida.
Garantikvittot innehåller 21 fält. En del av fälten är lätta att ta reda på eftersom det står på en stor lapp som hänger på cykeln. Men fältet för t.ex. återförsäljarnummer beror på vilket märke det är, på cykeln, och står inte på den lappen. Det tog ungefär 3,5 minuter att skriva ett garantikvitto med dagens system.

Även om vi inte kan få ner tiden kommer det bli betydligt mycket lättare vid arbetsgången med garantikvitton. Det är en fråga om bekvämlighet samt tydlighet. Eftersom större delen av garantikvittona skrivs för hand kan det variera med hur tydlig handstilen är.

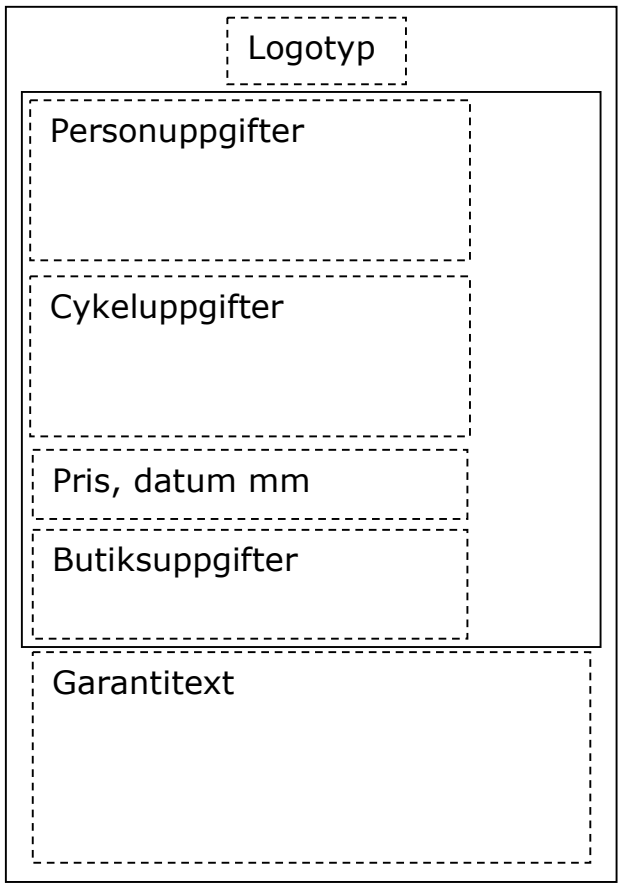
Figur 3 visar hur ett av dagens garantikvitton ser ut.


TEAM SPORTIA 029
BikePartner
Samtliga uppgifter är obligatoriska.
Personnr/Org.nr: 841028 4056
Telefon: 0733 – 248084
Förnamn: Anders
Efternamn: Bergh
Adress: Gustav Adolfs 13
Postnr: 25263
Ort: Helsingborg
E-mail: ihm03abe@student.lu.se
Hem/Försäkringsbolag:
Fabrikat: EXTREME SJÖSALA
Art.nr: 56721.5
Ramnr: D136173
Regnr: SEC 160631.39
Nyckelnr: 2231
Cykeltyp: Dam Herr Barn MTB Tandem Racer
Färg: Blå Brun Gul Grön Silver Vit
Ange endast en färg. Guld Lila Grå Röd Orange Svart
Cykelns pris: 4995 kr Extrautr: kr
Totalt: kr Avgår: kr
Summa: 4995 kr
SR Åfnr: 31618 Förs.datum: 26/4-2006
Ovanstående uppgifter registreras i S Reg AB och blir därmed även tillgängliga för polis, försäkringsbolag och cykelleverantör.
TEAM SPORTIA HÄSSLEHOLM
Kundens ex

Figur 3

3.2.2 Garantikvittots utseende

Det finns tre olika garantikvitton på Team Sportia – ett för varje märke. Vi har kollat mest på Extremes då det kvittot innehåller mest information. Men de andra kvittona har också funnits med i bilden när vi diskuterat med Team Sportia om design på kvittot. Extremes kvitto är uppdelat i grupper. I den första finns personuppgifterna och i den andra finns uppgifterna för cykeln. Men det upplägget blir väldigt kompakt och svårläst. Eftersom kvittot ska skrivas ut på en A4-skrivare väljer vi istället att placera all information under vartannat.



Figur 4

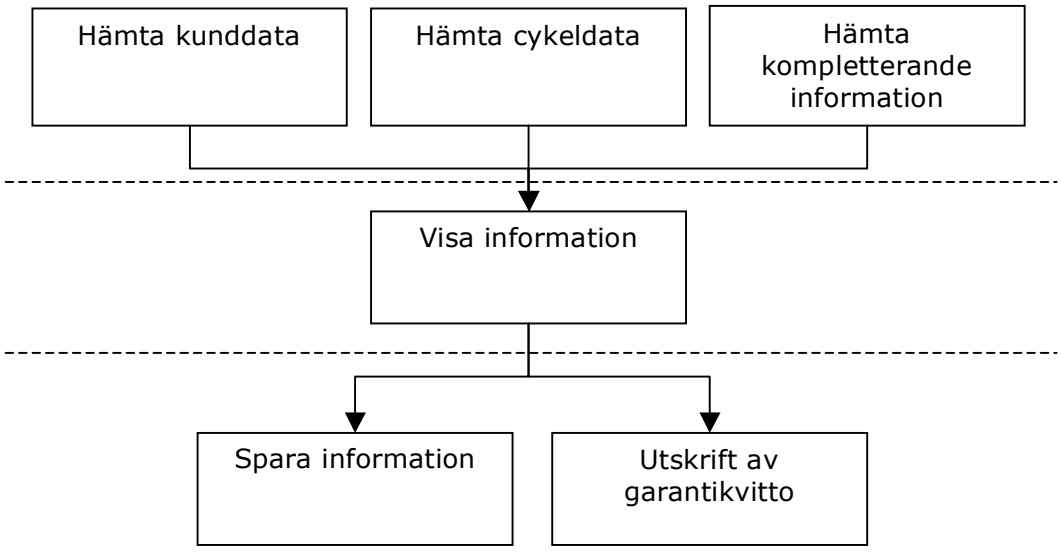
Figur 4 visar kvittots uppbyggnad. Först följer Team Sportias logotyp. Person- samt cykeluppgifter är två av textblocken. Sedan följer prisuppgifterna. Den rutan kan se olika ut beroende på om kampanjpris och tillbehör finns med. Garantitexten behöver ett stort utrymme och behöver antagligen skrivas i något mindre storlek för att få plats med all text. På det här sättet behöver man endast skriva ut ett A4 utan dubbelsidig utskrift.

3.2.3 Sammanfattning

Tiden det tar att registrera ett kvitto idag tar ca 3,5 minut. Det är inte bara tiden som är det viktiga utan hela arbetsgången är komplicerad idag. Handstil kan också påverka läsligheten av dagens kvitto. Layouten gör att kvittot blir lättare att läsa.

3.3 Specificerad funktionalitet

Applikationen vi skapar ska klara av att utföra ett flertal funktioner. Det ska gå att hämta in information. Det ska även gå att spara och skriva ut den hämtade informationen. Man kan dela upp tillvägagångssättet i tre delsteg. Första steget är att hämta informationen. Andra steget visar informationen som hämtats in. Det sista steget handlar om att skapa en utskrift av ett garantikvitto samt undansparande av informationen. Se figur 5 som visar vad applikationen ska åstadkomma. Bilaga C innehåller kravspecifikationen.



Figur 5

När det gäller att hämta kund- och cykeldata vet vi redan vad som ska finnas med på garantikvittot från Team Sportias sida. Men det finns information som saknas i databaserna som måste skrivas in i applikationen:

- Eventuellt kampanjpris – om kunden köper en cykel till kampanjpris måste ändå ordinarie pris finnas på kvittot eftersom man ändå ska vara berättigad till en cykel till det priset vid ett garantifall.
- Pris på eventuella tillbehör till cykeln – ifall att kunden köper till något extra till cykeln såsom lyse och lås ska det priset också ligga på kvittot.

- S-Regnummer – det här är ett speciellt försäkringsnummer som kan läsas in med en streckkodsläsare.
- Ramnummer – ramnumret på cykeln måste vara med och läses in med streckodsläsare.

Utöver den information som måste skrivas in finns det även information som beror på informationen i databaserna men som inte står i dem:

- Butiksinformation – namn på butik, kontaktinformation och butiksnummer
- Återförsäljarnummer – ett nummer som beror på vilket märke det är på cykeln. Det varierar mellan butikerna vilka cyklar de säljer.
- Garantitext – garantitexten beror på vilken återförsäljare det är. Det här är texten som ska stå på kvittot.

Den här informationen måste infogas på något sätt på garantikvittot. Det måste vara lätt att ändra eftersom det kan komma till nya märken eller ändringar på garantitexten framöver.

För kravspecifikation se bilaga F.

3.4 Skapa testmiljö

Vid skapandet av testmiljön avsåg vi att skapa något som låg så nära målmiljön som möjligt. Meningen var att minska antalet problem vid övergången samt att ha något att testa mot vid implementationen. Delstegen bestod i att installera Oracle- och IBM-databaserna.

3.4.1 Hemladdning av databasmjukvara

Tillvägagångssättet för att ladda ner databaserna var väldigt lika. För att få ladda ner gratisversionerna av databaserna behövde man först registrera sig på IBM respektive Oracles sajt. När väl registreringen var klar kunde man logga in och hämta de installationsfiler som man behövde.

3.4.1.1 Installation av IBM DB2 Express-C

IBMs installationsfil för DB2 Express-C låg på 309MB. Vid installationen var vägledningen bra. Det mesta var väl förklarat vilket gjorde det lätt att veta vilka val man skulle göra.

Vid installationen installerades en mängd verktyg vilket gjorde det svårt att veta vilket program man skulle använda för att skapa en databas. Programmet heter Styrcenter och nås under Startmenyn -> Alla program -> IBM DB2 ->

Administrationsverktyg. När man startar programmet har man möjlighet att välja om man ska ha en enkel vy eller en mera avancerad vy. Genom den enkla vyn har man tillräcklig funktionalitet för att skapa en databas. Efter val av vy är det enkelt att skapa en ny databas genom att högerklicka på Alla databaser och välja att man vill skapa en ny databas. Här skapade vi en databas som heter Customer.

Det var inte särskilt svårt att slutföra det här momentet. De inställningsmöjligheter som fanns var bra beskrivna i installationsprocessen och det tog inte lång tid att sätta sig in i hur klientprogramvaran fungerade. Det fanns en mängd inställningsmöjligheter för prestanda och minneshantering som vi inte konfigurerade men eftersom vi inte var ute efter prestanda utan bara själva tekniken hur man ansluter var det inget vi inriktade oss på.

3.4.1.2 Installation av Oracle 10g Express

Oracles installationsfil för 10g Express låg på 216MB. Installationen av databaserna var lätt eftersom installationsprogrammet gav bra vägledning. Installationen tog hela 1,3GB i anspråk. Det här kan ha berott på att vi valde ett paket vi inte behövde.

Oracles databas nås via ett webbgränssnitt. Webbgränssnittet hade få funktioner, men det gjorde det lätt att hitta i. Men här saknades funktionen att skapa en egen databas för endast produkterna. Därför skapades de kommande tabellerna i den databas som skapades vid installationen.

Om man jämför Oracles medföljande klientprogramvara mot IBMs lämnade det mycket att önska i funktionalitet. Det hade varit bra att kunna skapa en egen databas för endast produktdatan. Nu fick den läggas i systemdatabasen istället. Det här påverkar inte upplägget av applikationen alltför mycket eftersom det endast är ett annat namn på databasen vi behöver ansluta till vid övergången. Det hade med största säkerhet varit möjligt att skapa en ny databas i en annan klientprogramvara men det här gav oss ändå möjlighet att testa våra anslutningar.

3.4.2 Skapa databaser och deras tabeller i testdatabaserna

Vi skapade en databas i IBM-installationen och en i Oracle-installationen. IBM databasen skall innehålla kunder medan Oracle skulle innehåller alla produkter.

Tabellerna som skapades innehöll de kolumner som vi fick reda på vid undersökning av Team Sportia.

Eftersom Team Sportia inte visste om garantidatan skulle sparas i en IBM- eller Oracle-databas skapades tabellen i båda databaserna. Vi kom fram till att det inte skulle orsaka något större problem vid den kommande implementeringen. Det här skulle dessutom möjliggöra att Team Sportia kunde välja vilken databastyp de ville i ett senare skede.

3.4.3 Beskrivning av datorkonfiguration

Testmiljön kör Windows XP med Service Pack 2. Datorn, testmiljön ligger på, har en Intelprocessor med dubbla 1,83GHz-kärnor och har 256MB minne installerat.

Det som kan skilja sig gentemot målmiljön är att det är en annorlunda Windowsversion – antagligen en serverversion (Windows 2003 Server eller Windows Server 2000). Men varken Windowsversion eller hårdvara ska påverka applikationen negativt.

3.4.4 Sammanfattning

Testmiljön som vi skapat är så lik målmiljön som möjligt. Vi har skapat garantitabellen i båda databastyperna eftersom Team Sportia inte visste vilken av dem de skulle välja senare.
Installationen av båda databaserna gick relativt lätt.

3.5 Plattform för programmering

Val av programspråk och utvecklingsmiljö hänger tätt ihop. De är så tätt förankrade så att det är svårt att dela upp dem under två rubriker. Programspråk följs av utvecklingsmiljö men det kunde lika gärna ha varit tvärtom.

3.5.1 Val av utvecklingsmiljö

Av de paket som klarar av att skapa visuella gränssnitt samt koppla till databaser smidigt finns det två stora paket. Eftersom vi vill utveckla i en utvecklingsmiljö med bred kundbas, då det finns mer hjälp att hämta från forum och dokumentation, faller många mindre verktyg bort även om de hade fungerat. Av de två paket vi har kollat på kommer det ena från Borland och det andra från Microsoft. Båda klarar av våra krav på databaskopplingar och utskrifter. Ser man till priset ligger Microsofts paket på mindre än hälften av Borlands utvecklingsmiljö. Vi har kollat på Microsoft Visual Studio 2005 Standard samt Borland C++ Builder 2006 Professional. Om man bortser från priserna ser båda verktygen lika användbara ut.

Valet föll på Microsoft Visual Studio som hade den bästa kombinationen av funktionalitet och dokumentation. Det finns en mängd forum där man får snabb hjälp på sina problem. Microsofts egen webbplats MSDN har en hel mängd exempelkod och bra detaljerade förklaringar på hur klasser är uppbyggda och hur de fungerar. I Visual Studio finns det möjlighet att skapa databasanslutningar via ADO som är en teknik Microsoft utvecklat för att använda samma åtkomstfunktioner

oberoende av vad det är för databas. Med hjälp av Visual Studio går det lätt att skapa ett grafiskt gränssnitt som är lätt att ändra på vid behov.

3.5.2 Val av programspråk

Vi har tittat på några olika programspråk för att väga fördelar gentemot nackdelar. De webbaserade språk som HTML, PHP eller APS föll bort eftersom applikationen inte ska köras via en webbläsare. De språk vi tittade på var:

- Visual Basic
- C#
- C++
- Java

Utifrån de krav vi hade på att vi skulle kunna språket fanns bara Java, och C++ kvar att välja mellan. Anledningen till att Visual Basic inte gick vidare var för att vi använt det i många samanhang tidigare och vill använda oss av något vi lärt oss under utbildningen. När det gäller C# hade ingen av oss någon tidigare kunskap eller erfarenhet av detta språk och därför föll det bort, medan vi läst C++ som kurs terminen innan.
Det hade gått att utveckla i flera olika programmeringsspråk i Visual Studio, men det hade blivit betydligt svårare att sätta sig in i koden. Vi hade även varit tvungna att sätta oss in i hur kommunikationen mellan programmeringsspråk fungerar vilket hade lett till längre tid innan man kunde börja utveckla applikationen. Både Java och C++ är objektorienterade. Eftersom programmet bara skulle gå att använda på Windows behövdes inte Javas kompatibilitet mellan olika plattformar. Dessutom fanns det mera exempelkod inom C++ än Java på IBM och Oracles sajter. Vi bedömde att den här exempelkoden kunde vara värdefull vid implementeringen ifall att någonting inte fungerade. En annan faktor var att C++ är snabbare än Java vid körning av programmen.

Vi valde C++ eftersom det är ett språk vi känner till och att stödet är bra både från Oracle och IBM när det gäller exempelkod. Det är de två största argumenten varför vi valde språket.

3.5.3 Sammanfattning

Vårt val av programspråk föll på Visual C++ eftersom det är objektorienterat och är snabbare än Java. Det berodde även på att det fanns mera exempelkod och bättre forum för vårt ändamål. Användarvänlighet och dokumentation avgjorde vilken utvecklingsmiljö vi valde.

3.6 Val av utskrifts- och databasfunktioner

3.6.1 Utskriftsbibliotek

Det var ett krav att kunna skriva ut en garantiutskrift som innehåller den information garantikvittot innehåller idag. Det ska även gå att skriva ut utan att användaren får upp en dialogruta. Vi väljer att skriva ut garantikvittot på den skrivare som är satt som standardskrivare. En logotyp i form av en bitmapsbild och någon form av ram ska finnas med på utskriften utöver den ínformation som ska skrivas ut.

Vi har sökt efter olika klasser man kan använda för att skriva till en skrivare men de svar vi fått på forum eller genom att söka på Google har mestadels pekat på samma håll. I och med valet av Visual C++, som språk, följer den mest grundläggande klassen för presentation av grafik på en enhet automatiskt. CDC är en inbyggd klass i Visual Studio vilket innebär att man inte behöver inkludera några filer för att använda den. Det finns möjlighet att rita ramar samt skriva ut text. Det går även att hämta in vilken skrivare som är installerad som standard. Det här innebär att man slipper en dialogruta med inställningar var gång man väljer att skriva ut vilket sparar tid eftersom inställningarna på skrivaren alltid ska vara samma.

CDC fungerar oberoende av vad det är för enhet man skriver till. Det spelar alltså ingen roll om det är en skrivare eller bildskärm då det fungerar på samma sätt. Det hade alltså gått att skapa användargränssnittet på samma sätt om det inte hade funnits bättre vägar att gå för det.

När skrivaren ska väljas anropar man funktionen Attach. Innan dess har man hämtat vilken skrivare som är standard genom att skapa ett dialogobjekt som hämtar standardskrivaren.

För att kunna skapa layouten på kvittot behöver man ett antal funktioner. För att kunna skriva ut text behöver vi anropa en funktion som heter TextOut. Med hjälp av av den placeras även texten i rätt position.

Den svåraste biten är att hämta in en bitmap-bild som ska läggas på utskriften. Här måste man först och främst skapa ett bitmapobjekt som man sedan använder för att läsa in en bild till minnet. När man bilden är inläst i minnet ska man välja var på dokumenten bilden ska placeras.

3.6.2 Databasfunktioner

Det behövs en funktion att nå databaserna på som fungerar på båda databastyperna. Vi valde ADO (ActiveX Data Object) som är en teknik där man använder anslutningssträngar som är lätta att ändra. Det hade gått att använda OLE DB (Object Linking and Embedding for Databases) eller DAO (Data Access Objects) istället för anslutningen, men ADO är den senaste tekniken och

är mer lättförståelig att utnyttja i programmeringen. Tekniken är skapad för att slippa skriva olika kod beroende på vad det är för databastyp man ansluter till. Man använder ADO-anslutningssträngar för att skriva vilken databas man vill nå samt användarnamn och lösenord. Anslutningssträngarna är det som skiljer sig åt beroende på vilken databastyp man vill nå. För övrigt ser koden likadan ut för databasåtkomsten. För att sedan hämta informationen från databasen använder man sig av SQL-satser.

3.7 Implementering

Här beskrivs hur vi implementerade vår prototypapplikation. För att överhuvudtaget komma igång behövde vi ta reda på allt som applikationen skulle klara av. Därefter skapa klasser som gav en bra överblick av applikationens huvuddelar. Grundskelettet av klasser kompletterades sedan med kod för att uppfylla sin funktion.

3.7.1 Skapa klasser

För att komma fram till vilka klasser som skulle finnas med gick vi igenom vad programmet skulle klara av att utföra. Utifrån de uppgifter vi diskuterat fram med Team Sportia skulle programmet klara av följande funktioner:

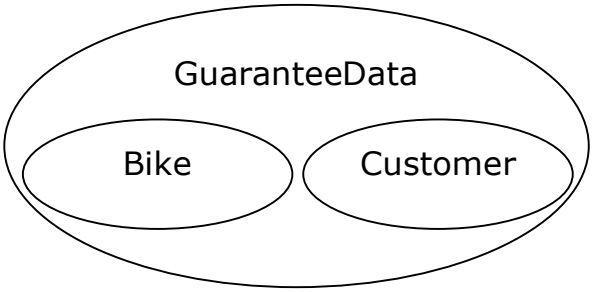
1. Hämta från produktdatabas (Oracle)
2. Hämta från kunddatabas (IBM)
3. Utskrift av garantipapper
4. Spara garantidata i databas (Oracle och IBM)
5. Visa gränssnitt för användaren

För varje funktion som behövdes skapades en klass som innehöll de metoder som krävdes för att objekten som skapas skall utföra sin uppgift: Numren ovan och nedan hör ihop. Vi skapade:

1. BikeDB
2. CustomerDB
3. PrintGuaranteeData
4. SaveGuaranteeData
5. CykelregistreringDlg

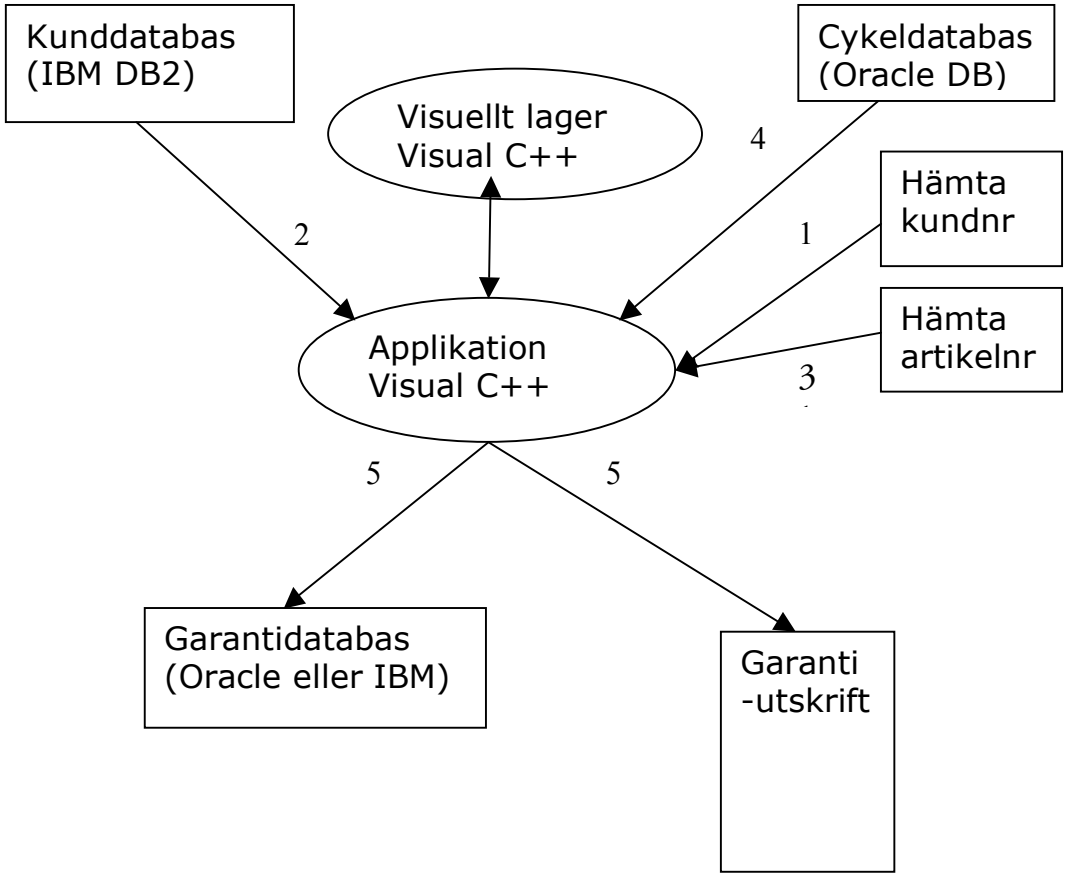
Det behövdes även en ”databärare” för att möjliggöra kommunikation mellan klasserna Grundfunktionerna höll upp strukturen men det saknades objekt för den information som skulle skickas emellan. Därför skapades klasserna Bike, för cykel, samt Customer, för kund. Men eftersom en cykel och en kund inte är allt

när det gäller garantidata skapades även en klass som hette GuaranteeData som fungerade som ett nav med information. GuaranteeData innehåller både Bike och Customer men även information som varken kan knytas till cykel eller kund. Ett exempel är datum för köp något som varken tillhör cykel eller kund men tillhör garantidata. Se figur 6.



Figur 6

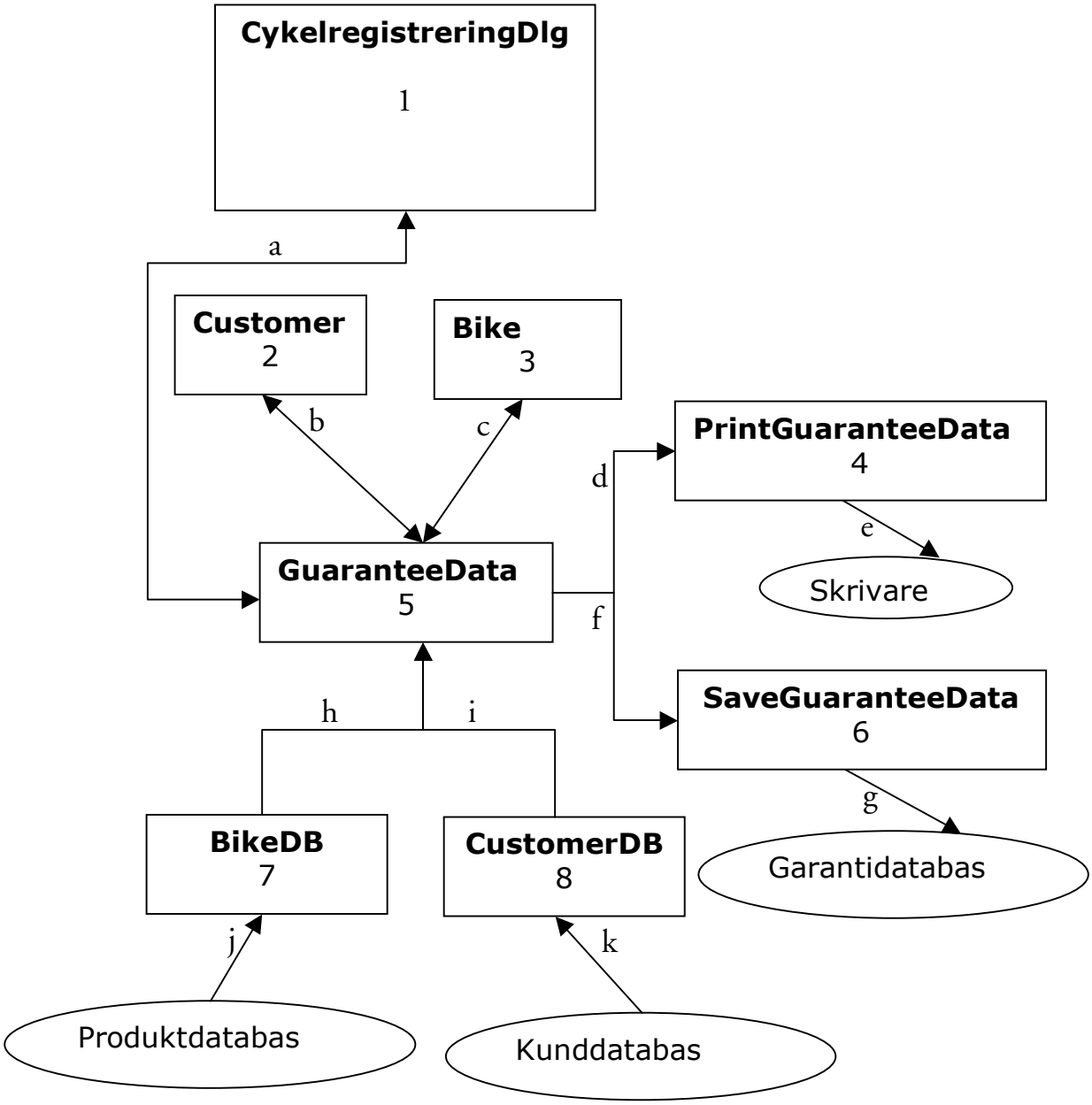
Vi valde den här uppdelningen delvis för att man då kunde ändra en enskild klass utan att påverka funktionen hos någon annan. Det här underlättade även genom att man inte var beroende av att ha implementerat allt för att testa delar av applikationen.



Figur 7

Figur 7 visar programmets uppbyggnad.

Applikationen hämtar först in (1)kundnummer. Den informationen används sedan för att hämta alla uppgifter om kunden från (2)IBM databasen. Sedan hämtar programmet in (3)EAN-nummret som används för att hämta all produktdata från (4)Oracle databasen. Sedan sparas den hämtade data inkl tilllägsfält i (5)garantidatabaserna samt ett garantipapper skapas och (5)skrivs ut. Det visuella lagret ligger bara ovanpå för att kommunicera med användaren



Figur 8

Figur 8 visar hur klasserna hör samman och hur de kommunicerar. GuaranteeData är navet och har all information som behövs för att skapa ett garantikvitto. CustomerDB och BikeDB hämtar in kund- samt cykeldata till GuaranteeData. Customer och Bike är byggstenar till GuaranteeData.

3.7.2 Kronologist körschema

När man startar programmet ser man gränssnittet som styrs av CykelregistreringDlg(1). Där fyller man i personnummer som skickas till GuaranteeData(5). GuaranteeData(5) anropar CustomerDB(8) som i sin tur retunerar all information om kunden. GuaranteeData(5) skickar denna information till Customer(2) där den lagras. Samma information skickas även tillbaka till CykelregistreringDlg(1) för att visas i kundinformationsfälten i programmets gränssnitt.
Nästa inmatning är EAN-numret som skickas från CykelregistreringDlg(1) till GuaranteeData(5) som i sin tur anropar BikeDB(7) med EAN-numret. BikeDB(7) hämtar cykelinformationen från produktdatabasen och returnerar detta till GuaranteeData(5) som skickar informationen till Bike(3) där den lagras för senare användning. GuaranteeData(5) skickar även den inhämtade informationen tillbaka till CykelregistreringsDlg(1) som visar informationen i cykelfälten i gränssnittet.
När all information är hämtad och lagrad i minnet(2, 3) klickar användaren på utskriftsknappen i gränssnittet(1). CykelregisteringDlg(5) anropar GuaranteeData(5) och funktionen för att skriva ut. GuaranteeData(5) hämtar då in den lagrade informationen från Bike(3) och Customer(2) och skickar den till PrintGuaranteeData(4). PrintGuaranteeData(4) behandlar då informationen och lägger in den i det förprogrammerade layouten för utskrift. Samtidigt som PrintGuaranteeData(4) anropas, anropas även SaveGuaranteeData(6) där all information sparas till en databas.

3.7.2.1 GuaranteeData

Klassen ligger i centrum och innehåller all garantidata som består av en Bike (cykel), en Customer (kund) och återförsäljarnummer. Bike och Customer är skilda åt eftersom de hämtas från två olika databaser. I GuaranteeData finns det funktioner för att returnera en Bike och en Customer. Det finns även funktioner som anropar BikeDB och CustomerDB för att hämta respektive data.

I GuaranteeData hämtas butikens uppgifter, garantitext och återförsäljarnummer in från en fil. Den här informationen ska stå på garantikvittot och är specifik för respektive butik. En annan möjlighet hade varit att skapa en databas med den här informationen men eftersom vi inte vill blanda in ännu en databasanslutning i programmet väljer vi bort den lösningen.

Det finns även en funktion som returnerar datum så att kvittot och informationen som ska sparas får en datumstämpel.

3.7.2.2 CykelregistreringDlg

Klassen har hand om kommunikationen mellan Användargränssnitt och GuaranteeData. Här ligger funktioner som hämtar in data från textfälten i dialogrutan och sedan skickar det vidare till GuaranteeDatas funktioner.
Exempel: När man skrivit in personnummer för kunden trycker man på ”Hämta personuppgifter”. Då anropas en funktion i GuaranteeData som hämtar data från IBM-databasen.
Använder man en streckkodsläsare avslutar den med ett enterslag vilket gör att man inte behöver trycka på knappen manuellt. Speciella drivrutiner för streckkodsläsare behövs inte.

3.7.2.3 Customer

Klassen har hand om kunddata såsom personnummer, adressuppgifter och namn. Genom funktioner sparas och hämtas datan som lagrats i GuaranteeData man skapar.

3.7.2.4 Bike

Klassen fungerar på samma sätt som Customer men istället lagras produktdata såsom märke och artikelnr. Här finns en funktion som hämtar återförsäljarnummer som ska stå med på garantikvittot.

3.7.2.5 CustomerDB

Klassen ansluter till IBM-databasen för att hämta kunddata. Den anropas från GuaranteeData.
Personnumret skickas till klassen som en CString och informationen som hämtats retururneras som ett Customer-objekt till GuaranteeData.

3.7.2.6 BikeDB

Klassen ansluter till Oracle-databasen och anropas från GuaranteeData. Här hämtas produktdata. EAN-koden skickas till klassen som en CString och informationen som hämtats returneras som ett Bike-objekt till GuaranteeData.

3.7.2.7 PrintGuaranteeData

Klassen skriver ut information utifrån de data som ligger i GuaranteeData. Den anropas från CykelregistreringDlg. Indata är ett GuaranteeData-objekt.

3.7.2.8 SaveGuaranteeData

Klassen innehåller två funktioner för att spara garantidata i en databas – en för varje databastyp. Eftersom Team Sportia inte visste vilken databas som anslutningen skulle ske till valde vi att implementera för båda typerna då det egentligen inte var särskilt mycket merjobb. Indata för den här klassen är ett GuaranteeData-objekt.

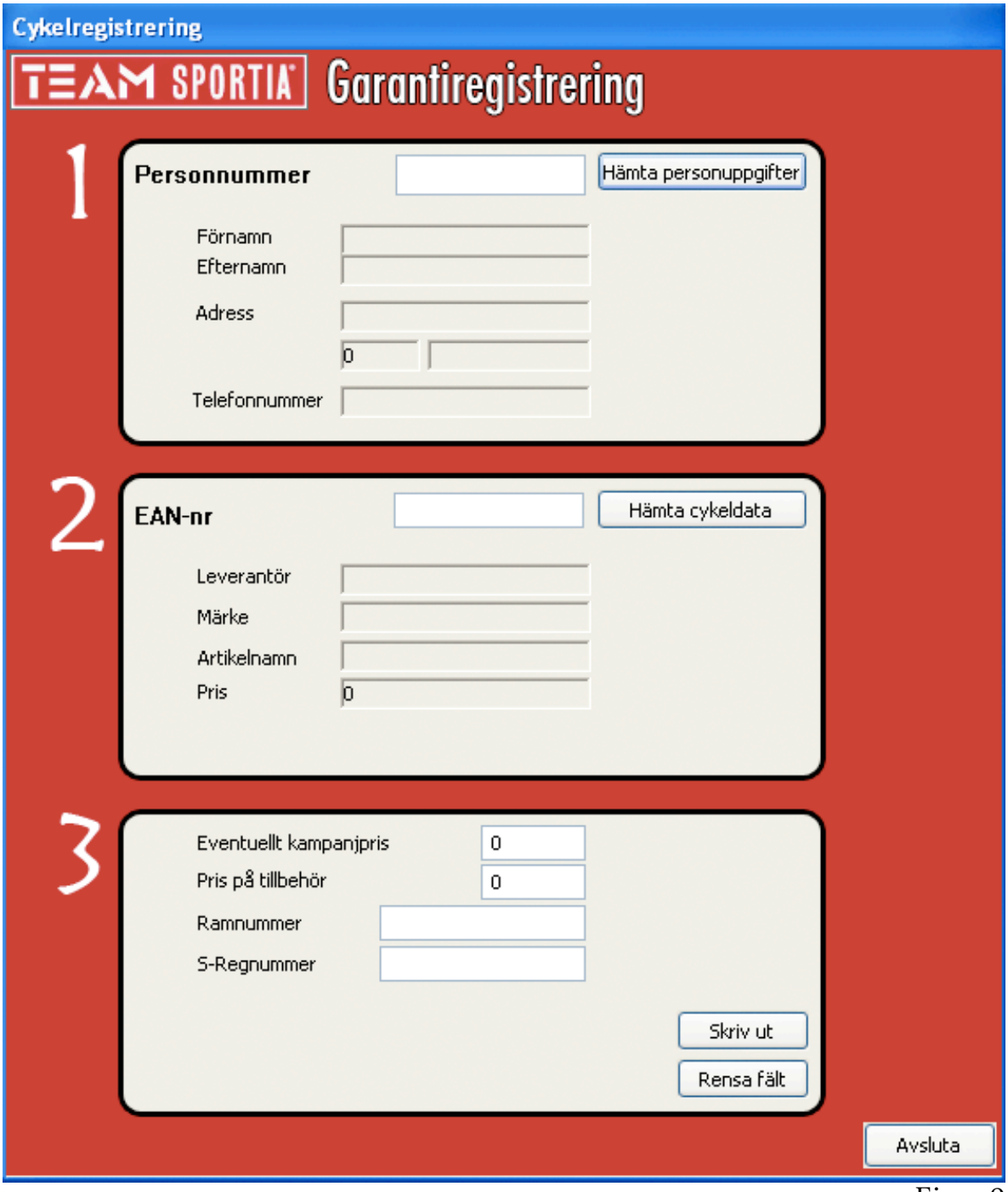
SaveGuaranteeData är implementerad på samma sätt som BikeDB och CustomerDB. Det som skiljer är vilka fält som skickas med. Den största skillnaden gentemot de två klasserna som hämtar information från databaserna är att den här klassen inte skickar någon information tillbaka. Det behövs inte på grund av eventuella felmeddelanden skickas direkt från klassen.

3.7.3 Skapa ett visuellt gränssnitt

När vi implementerat all funktionalitet till programmet behövde vi ett gränssnitt för att på ett enklare sätt kommunicera mellan användaren och funktionerna. Det här underlättar alltså för användaren.

Applikationen är tänkt att fungera i följande steg:
1. Läsa in personnummer, med streckkortsläsare eller tangentbord, från kundkort.
2. Trycka på knappen ”Hämta personuppgifter”.
3. Läsa in EAN-kod från streckkoden på cykeln.
4. Trycka på knappen ”Hämta cykeldata”.
5. Skriva in ramnummer samt S-regnummer.
6. Skriva in eventuellt kampanjpris och pris på tillbehör.
7. Trycka på knappen ”Skriv ut”.

I figur 9 visas dialogrutan för applikationen. Det här är den enda som visas utåt mot användaren.

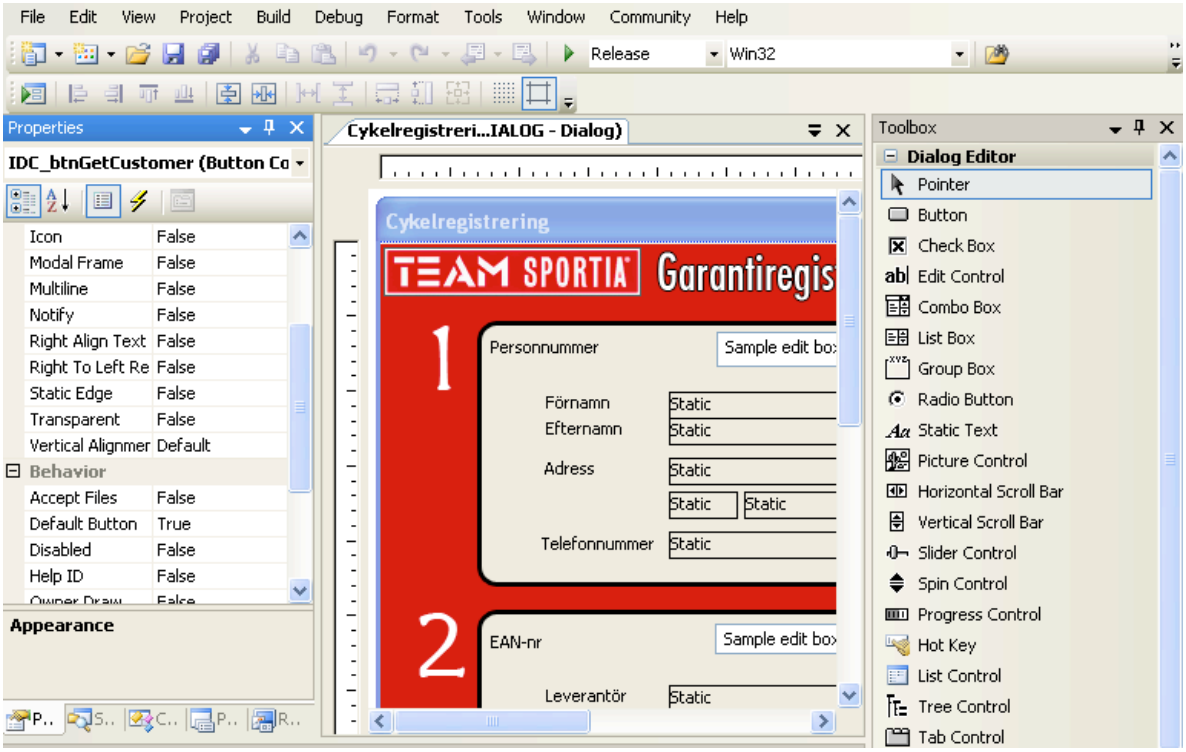


Figur 9

När information har hämtats från databasen hoppar markören till nästa fält för att man ska slippa att själv markera det. Något vi inte har lyckats med är att ändra vilken knapp som ska vara standard när man trycker enter. Det går att ändra i menyerna när man skapar en knapp men det går inte att ändra i ren kod. Nu är knappen ”Hämta personuppgifter” standard men annars hade man kunnat ändra standardknapp beroende på vilket fält man är på. Exempelvis skulle knappen ”Hämta cykeldata” markeras när markören var i fältet ”EAN-nr”. När att man måste tabba eller flytta muspekaren till rätt knapp.

Gränssnittet skapades från Resource View i Visual Studio. Med hjälp av verktyget underlättades arbetet med programmets utseende eftersom man inte behövde programmera för att skapa knappar eller textrutor. Istället placerade man knappar och textfält på arbetsytan. De fält som skulle fungera som input skapades som Edit Conrol. Outputfälten skapades som Static Text eftersom man inte skulle skriva i dem. Static Text användes även för att skriva rubriker. Knapparna för att skriva ut och hämta data använde vanliga knappar.

Figur 10 visar utvecklingsmiljön. Till vänster finns inställningar för den knapp eller det fält man markerat och till höger finns de knappar och fält man kan dra in



Figur 10

3.7.4 Hämta och spara data i databaserna

Anslutningssträngarna skapades genom att först och främst hämta information från en sida på internet med anslutningssträngar [6]. Men de modifierades, med korrekta lösenord och användarnamn samt tillägg från Visual Studios databaskopplare, för att fungera.

För att förenkla vid flytten till Team Sportias system valde vi att lägga anslutningssträngen utanför programmet. Den hämtas in när man vill ansluta till databasen istället. Detta ger fördelen att man inte behöver kompilera om var gång något i strängen ändras. Det kan t.ex. vara ett tillfälligt fel på servern och då går det att styra om anslutningen till en speglad server under tiden. Användarnamn och lösenord skiljer sig mellan butikerna och då ska man inte behöva kompilera en version till varje butik.

3.7.4.1 Kunddatabasen (IBM)

Filen för anslutningssträngen heter customerdb.txt.

3.7.4.2 Produktdatabasen (Oracle)

Filen för anslutningssträngen heter productdb.txt.

3.7.4.3 Garantidatabasen (IBM och Oracle)

Filerna för anslutningssträngarna heter oraguaranteedb.txt och ibmguaranteedb.txt. De tre första bokstäverna står för vilken databastyp man ansluter till. Strängarna man skapar ser likadana ut som för produkt- och kund-databasen

3.7.5 Skapa utskrift

För att skapa ett kvitto som är dynamiskt skapade vi variabler för radmellanrum och tabblängder. På det viset är det lätt att modifiera kvittot i efterhand. Istället för att ändra varje funktion, som skriver ut något, behöver man endast ändra några få variabler. Vi skapade även variabler för de olika fontstorlekarna av samma anledning.
Variablerna är skapade i koden istället för i konfigurationsfiler. Nu måste man kompilera om var gång man vill ändra utseendet på kvittot. Vi bedömer att man inte kommer att ändra utseendet på kvittot när det väl är satt. Kvittot ska se likadant ut i alla butiker.

Kampanjpris och pris på tillbehör skrivs endast ut ifall att de är angivna i applikationen. Det här var inget direkt krav från Team Sportia men det var inget svårt att implementera och gör det betydligt lättare för kunden att läsa.

Figur 11 visar kvittot som programmerades.

TEAM SPORTIA

Garantikvitto

Ägare:	Daniel Nord
Personnummer:	198304083598
Adress:	Snapphanegatan 10B 28136 Hässleholm
Telefon:	0451-84413
Märke:	Extreme Zoom
Tillverkare:	Team
Återförsäljarnr:	T12345-34
Ramnummer:	34534634646
S-Reg.Nummer:	S46346
Pris:	1499
Kampanjpris:	1200
Tillbehörspris:	200
Summa:	1400
Försäljningsdatum:	2006-06-01
Säljare:	Team Sportia Hässleholm Kvantumhuset 0451-38 56 50 HLM1234

Ovanstående uppgifter registreras i S Reg och blir därmed även tillgängliga för polis, försäkringsbolag och cykelleverantör.

1. Garanti från försäljningsdatum avseende fabrikations och materialfel som har förelegat vid leveransdatum
2. 1 Månads utökad reklamationsrätt för alla Garantiformer
3. Garantin omfattar EJ fel som uppstått genom yttre påverkan, ovarsamhet, normalt slitage felaktigt hanterande och liknande hanteringssätt.
4. Garantin omfattar Ej rese och transportkostnad.
5. Vid åberopande av garantin måste alltid kvitto eller faktura bifogas. Sänd / bifoga helst en KOPIA på kvittot eller frakturan och behåll originalet.
7. För en snabbare behandling av garanti / reklamation kontakta alltid oss innan du sänder in varan

Figur 11

3.7.6 Sammafattning

Implementeringen var den största fasen och här skapades klasser utifrån applikationens kommande funktioner. Det behövdes även tre klasser för att hålla reda på informationen innan den hade skrivits ut eller sparats i databasen.

Vi delade upp projektet i dess grundfunktioner. Utifrån dem skapade vi klasser. Klassen GuaranteeData blev i centrum för all kommunikation. Det är den som anropar funktioner för att hämta data eller skriva ut.

Gränssnittet skapades från Resource View i Visual Studio. Genom att positionera knappar och textfält visuellt var det enkelt att snabbt skapa ett fungerande gränssnitt.

Applikationen vi skapat klarar våra krav utifrån problemformuleringen. Den klarar av att ansluta till vårt testsystem och hämta data därifrån.

3.8 Test på målsystem

Vi testade hur applikationen fungerade på en nyformaterad dator och på Team Sportias datorer. Vi testade vilka filer som behövs för att överhuvudtaget starta applikationen. Vidare skulle vi testa att ansluta till databaserna men här stötte vi på stora problem på grund av att den information vi fått av Team Sportia inte stämde.

3.8.1 Filer man behöver för att få igång applikationen

För att vara säker på att så få komplikationer som möjligt uppstod formaterade vi en dator och installerade Windows XP på nytt. Detta för att vara säker på applikationen fungerade utan speciella DLL-filer. Men programmet fungerade inte alls. Man fick en dialogruta där man informerades om att programmet kunde vara felkompilerat eller att det saknade dll-filer. Eftersom allt fungerade på den Windows-installation som hade Visual Studio installerat trodde vi att det berodde på Microsoft .NET framework saknades. Men efter att ha installerat både version 1.1 och 2.0 utan något lyckat resultat var det något annat som behövdes för att kunna starta applikationen.
Med hjälp av information på olika forum kunde vi hitta en länk till ett kodbibliotek som heter Microsoft Visual C++ 2005 Redistributable Package. För att kunna installera det paketet behövde vi först installera Windows Installer 3.1.

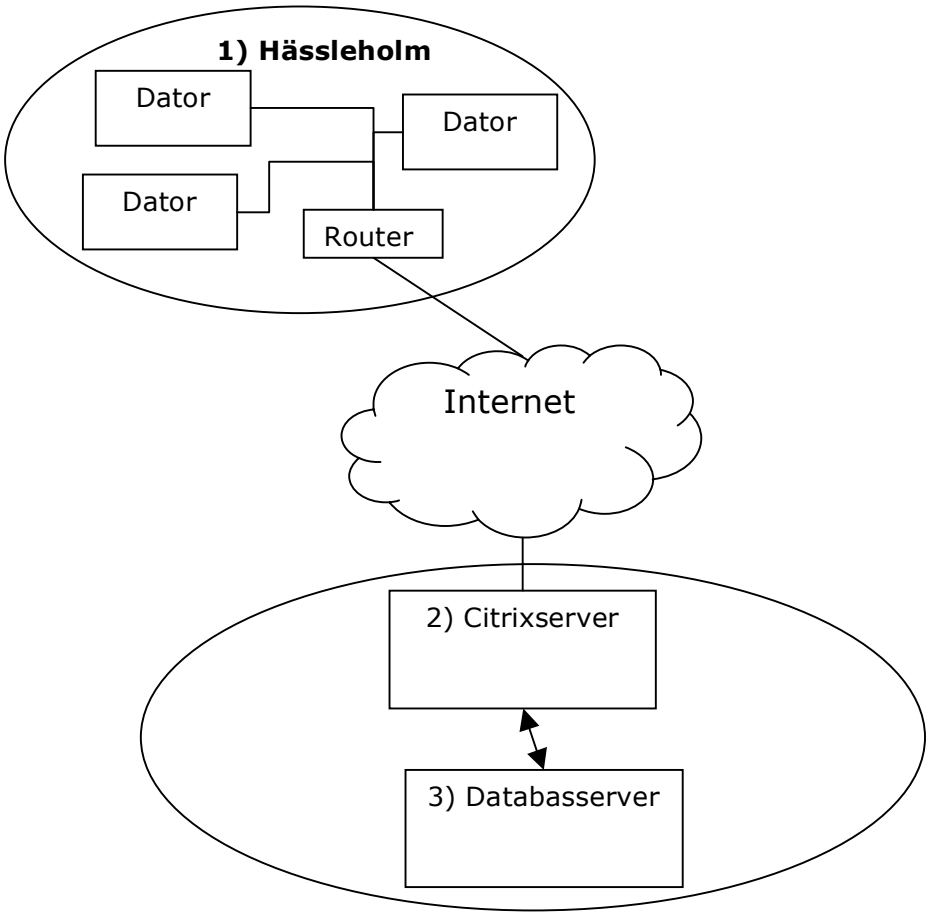
Vi testade vår applikation på både Windows XP Service Pack 1 och Windows XP Service Pack 2 för att försäkra oss om att det fungerar i båda versionerna.

Sammanfattningsvis behöver man installera Windows Installer 3.1 och Visual C++ 2005 Redistributable Package för att få applikationen att fungera.

3.8.2 Hämta kund och produktinformation

Applikationen behöver tillgång till dll-filer som installeras samtidigt som man installerar IBMs databas eller klientprogramvara. Det här fanns inte på de installationer som Team Sportia har vilket medförde att vi inte kunde testa den här funktionaliteten på det sätt vi tänkt. Men eftersom vi hade drivrutinerna på vår egen testdator testade vi att ansluta den till Team Sportias nätverk och därifrån försöka ansluta till databaserna. Efter att ha konfigurerat anslutningssträngarna gick det inte att ansluta mot databasservrarna ändå eftersom, vi vid testtillfället fick vi reda på att det inte går att nå dem över internet. Det här var något vi blev väldigt överraskade av eftersom vi hela tiden innan fått reda på att det ska fungera att ansluta på det sättet.
Eftersom vi ändå inte kunde ansluta mot databaserna över internet gick vi inte vidare till att installera IBM och Oracles klientdrivrutiner på Team Sportias dator.

Skillnaden, mellan det vi fick reda på från början och så det fungerar i praktiken, är att Team Sportia använder Citrix för att nå databaserna på servern. Figur 12 visar hur Citrix är uppbyggt. I Hässleholm finns ett flertal datorer som har en Citrixklient installerad. Klienten (1) ansluter över internet till Citrixservern (2). Citrixservern i sin tur har själva programmet som ansluter till databasservern (3). Programmet körs alltså på Citrixservern och visas sedan som en skärmbild på klienten i Hässleholm. Det här innebär att vår applikation, teoretiskt, skulle fungera på servern eftersom Citrix- och databasservern befinner sig på samma nätverk. De DLL-filer man behövt på klientdatorn tidigare behövs även här.



Figur 12

3.8.3 Utskrift från PrintGuaranteeData

När vi använde vår version av applikationen som klarar att skriva ut dummytext blev utskriften precis som väntat. Skrivaren som var satt som standard på testdatorn hämtades in utan problem. Det enda egentliga problemet vi hade, med utskriftsklassen var att typsnittet blev väldigt litet vid första utskriften. Men efter att ha ändrat och kompilerat om koden blev utskriften i rätt storlek. Av förståeliga skäl fungerade inte programmet med databaskopplingarna. Utskriften fungerade men datafälten blev tomma.

3.8.4 Sammanfattning

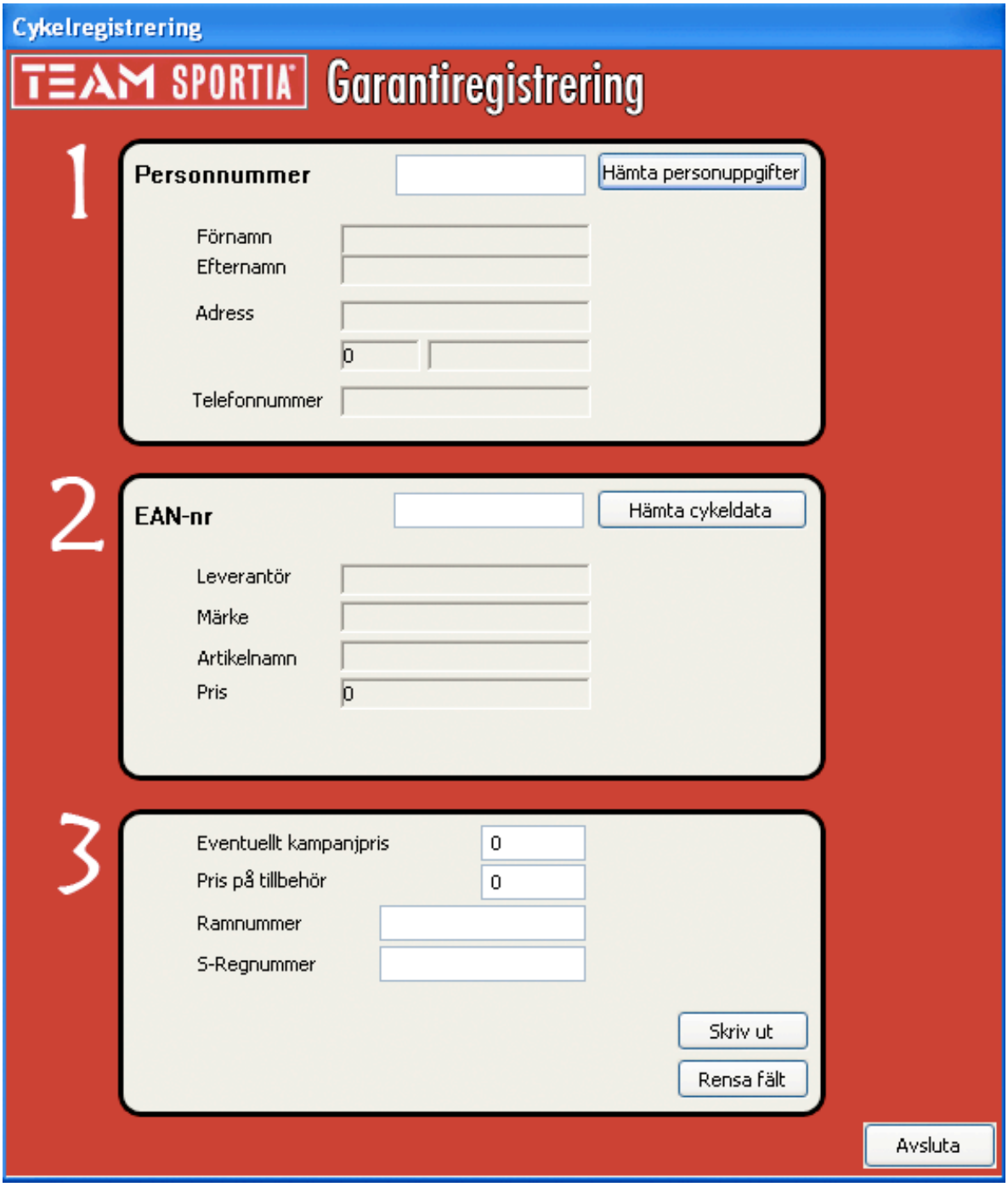
Vårt test på Team Sportia visade en hel del brister i applikationen. Den gick inte att starta till att börja med utan behövde kompletterande installationspaket för att kunna starta, se bilaga C. Databasanslutningarna fungerade inte överhuvudtaget eftersom anslutning mot databaserna, över internet, inte fungerade eftersom Team Sportia använder sig av Citrix, en teknik att nå en servers programvara på en klientdator. Utskriften visade sig dock att fungera utan problem.

3.9 Utvärdering

Under utvärderingen testade vi användbarheten på Team Sportias anställda och rättade till brister på gränssnittet som uppkom under testet. Testet utfördes på vår bärbara utvecklingsdator efter och under implementeringsfasen. På den datorn fanns även databaserna så att applikationen låg så nära den färdiga som möjligt.

3.9.1 Test av gränssnitt

Vi testade användbarheten på Team Sportias anställda eftersom det är de som är målgruppen.



Figur 13

Figur 13 visar vårt gränssnitt som inte har många fält att fylla i. Vårt test visade att det var lätt att sätta sig in i hur det skulle fungera. Eftersom de fält som går att fylla i är vita och resten av fälten inte går att klicka i ser man tydligt var man ska skriva in uppgifter. Eftersom vi testade när vi hade skapat ett gränssnitt i Visual Studio blev testet så verklighetsnära som möjligt. Det här underlättade eftersom gränssnittet var på skärmen istället för t.ex. en pappersprototyp.

Användarvänligheten har ökat om man jämför med att behöva skriva ett garantikvitto för hand eller i Excel, som man gjorde tidigare. Det är färre fält som

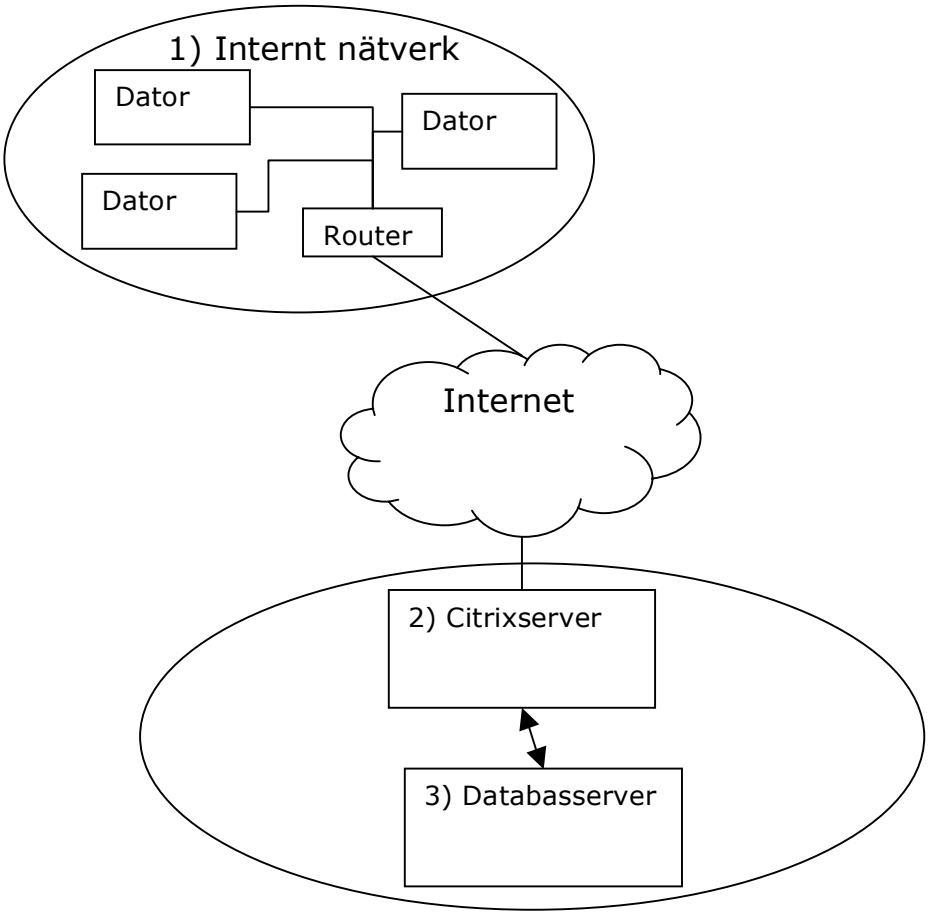
kan bli fel eftersom merparten av informationen kommer från databasen. De fält man skriver in manuellt kan bli fel inskrivna. Men skriver man fel personnummer eller EAN-kod kommer informationen inte att hittas alternativt hämtas fel person eller cykel. Den information som hämtas kan jämföras med informationen på cykeln och gäller det personens namn och adress kan man alltid fråga kunden.

4 Citrixproblematik

Den här punkten har kommit till på grund av att vi fick felaktig information från Team Sportia i början. Problemen är beskrivna i genomförandet men vi vill även försöka reda ut vad för problem det här kommer att leda till och hur Citrix fungerar för att visa att det förmodligen skulle fungera att föra över vår applikation till Citrix-servern. Hade Team Sportias systemutvecklare hunnit med att hjälpa oss att med ett konto att testa på Citrixservern hade vi försökt starta vårt program och anpassa eventuella inställningar. Men eftersom Team Sportia i dagsläget har sjösatt en hel del nya tekniker den här våren har systemutvecklarna haft alltför mycket att göra.

4.1 Citrix uppbyggnad

Citrix är en lösning skapad för att köra program från en server. Figur 14 visar uppbyggnaden. Istället för att programmen exekveras på klienten på det interna nätverket (1) körs de från servern (2). På klienten visas istället bilden av hur programmet ser ut. Tangenttryckningar samt musklick skickas till servern. För användaren ser det ut som att programmet körs på klienten i nätverket men i programmet ser det som att det körs på servern.



Figur 14

I Team Sportias fall körs försäljningsapplikationer och databaser från Citrix vilket medför att säkerheten är högre. Det går inte att nå databaserna utifrån förutom genom Citrix-klienten.

Varje butik har en s.k. hemmamapp, på Citrix-servern, där just deras information lagras. Här kan man t.ex. lagra konfigurationer för program som körs eller dokument som endast hör till butiken.

4.2 Installation av program

För att köra ett program via från en Citrix-server behöver man installera programmet, på servern, precis som på en vanlig arbetsstation. Det som skiljer är att man ska välja vilka som får ha tillgång till applikationen. För vår del innebär det här att vi kommer att behöva lägga konfigurationsfilerna i butikens hemmamapp. Dessa ligger i en underkatalog till programmet.

4.3 Fördelar med Citrix

Det finns två stora fördelar med Citrix. För det första är det säkerheten som ökas eftersom den enda port som behöver vara öppen är porten för Citrixklienten. För det andra är det betydligt lättare att administrera systemet eftersom endast en server behöver uppgraderas istället för ett flertal datorer. En annan fördel är att man kan köra flertalet program oberoende av operativsystem eftersom klienten finns för en mängd plattformar.

För Team Sportias del kan det vara en fördel att ha applikationen liggande på Citrix-servern eftersom man inte behöver tänka på anslutningen över internet till databaserna. Det blir även mycket lättare om applikationen ska användas i flera butiker eftersom man inte kommer att behöva konfigurera om anslutningsfilerna.

4.4 Nackdelar med Citrix

Om servern går ner beroende på t.ex. fel på hårdvaran eller någon form av angrepp blir alla klienter drabbade. Då gäller det att det finns någon form av backupserver.

Citrix lämpar sig bäst till kontorsapplikationer som t.ex. Microsoft Office-paketet eller andra program med lite rörlig grafik. Men det här är inget som kommer att påverka vår applikation.

4.5 Vad behöver anpassas till Citrix?

Det är på två ställen, i applikationen, man kommer att behöva ändra för att få den att fungera i en Citrixmiljö. För det första måste vi ändra sökvägarna till konfigurationsfilerna eftersom de inte kan ligga i samma mapp som applikationen då det inte skulle gå att ha olika butiksuppgifter på utskrifterna. För det andra måste vi ändra anslutningssträngarna så att de pekar på rätt databaser. Eftersom vi anslutit lokalt mot vårt testsystem, tidigare, skiljer sig inte anslutningsträngarna på utvecklingsversionen och den version vi skulle ha skapat för Citrix-lösningen. Sedan behöver IT-teknikerna skapa rättigheter i Citrix för applikationen så att Hässleholmsbutiken får köra den.

5 Beskrivning av programmet och dess kringmiljö

Den här delen beskriver hur vårt prototypprogram fungerar samt ger en beskrivning av den testmiljö vi använt under utvecklingen.

5.1 Prototypprogrammet

Applikationen är avsedd att utföra två saker. Det är att skriva ut ett garantikvitto och att spara garantiinformationen i en databas. Garantikvittot ska ges till kunden och informationen som sparas i databasen är butikens exemplar av kvittot.

Personnummer samt EAN-nummer matas in för att hitta rätt kund och cykel. För att kvittot ska bli fullständigt behöver man även komplettera med eventuellt kampanjpris och pris på tillbehör.

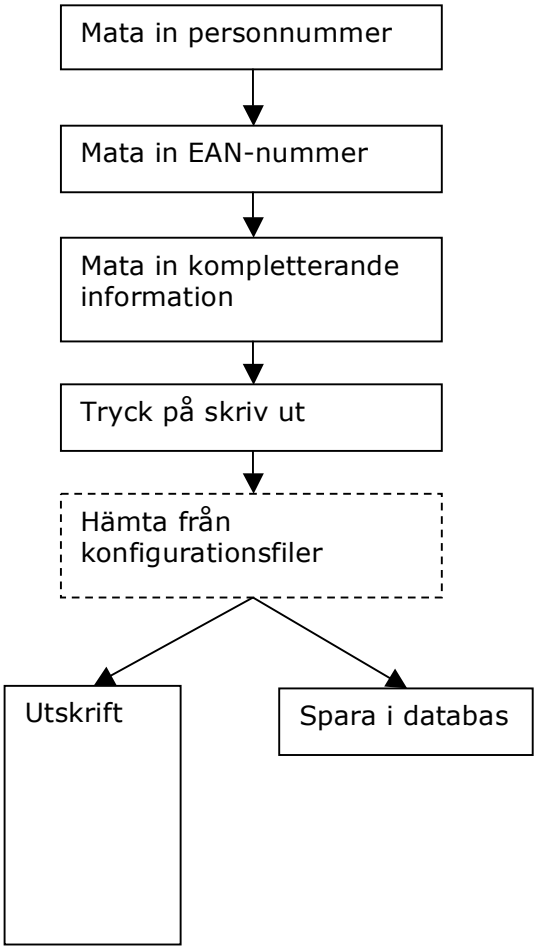
När man trycker på ”Skriv ut” i applikationen anropas ett flertal funktioner. Cykel- och kundinformationen hämtas in och slås ihop tillsammans med eventuellt kampanjpris och tillbehör. Återförsäljarnummer samt butikens namn, adress och butiksnummer hämtas från konfigurationsfiler. All den här informationen skrivs ut på kvittot som ska ges till kunden och även garantidatabasen får större delen av informationen inskriven.

När kvittot skrivits ut är även informationen sparad i databaserna. Vill man skriva ut en extra kopia är det bara att trycka på skriv ut igen. När man vill skriva in en ny kund trycker man på knappen rensa för att tömma alla fälten.

Där applikationen ligger finns en underkatalog, config, med konfigurationsfiler samt den bild som garantikvittot innehåller. All information som ska gå att ändra såsom adressuppgifter till butiken och garantitext finns här. Konfigurationsfilerna innehåller alltså den information som är individuell beroende på vilken butik man kör applikationen ifrån.

Om man skriver in ett felaktigt personnummer eller EAN-nummer som inte finns i databaserna kommer man få ett felmeddelande där det står att informationen inte går att hitta. Informationen som visas är felmeddelandet från ADO-anslutningen. Felmeddelandet ger information om var felet ligger så att man vet ifall att man matat in ett felaktigt värde eller om det bara är databaserna som inte går att nå.

Figur 15 visar tillvägagångssättet för applikationen.



Figur 15

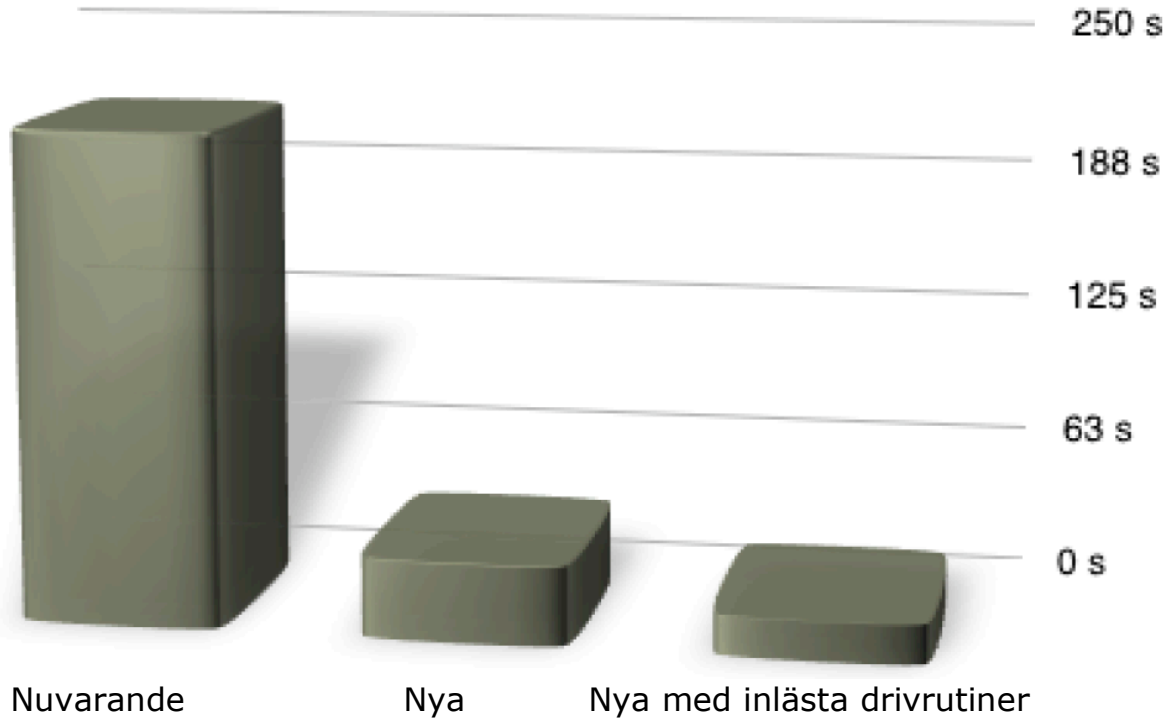
5.2 Specifikation av utvecklingsmiljön

På utvecklingsmiljön finns Oracle och IBMs databaser installerade. Testmiljön kör Windows XP med Service Pack 2. Datorn, testmiljön ligger på, har en Intelprocessor med dubbla 1,83GHz-kärnor och har 256MB minne installerat.

Det som kan skilja sig gentemot målmiljön är att det är en annorlunda Windowsversion – antagligen en serverversion. Processor och minne skiljer sig med största sannolikhet också. Men varken Windowsversion eller hårdvara ska påverka applikationen negativt.

6 Resultat

När vi mätte tiden för registreringsförfarandet av den gamla kvittotypen blev den ca 3,5 minut. På det nya systemet tar det drygt 30 sekunder att skriva ett garantikvitto när drivrutinerna för databaserna inte har laddats in. Vid andra körningen tar det knappt 20 sekunder att skriva kvittot vilket beror på att drivrutinerna för databaserna tid på sig för att laddas till minnet första gången. Det här förutser att inga problem sker under tiden. Se figur 16.



Figur 16

Vårt syfte var att skapa ett hjälpmedel för att underlätta garantiförfarandet vid försäljning av cyklar. Det har vi lyckats med om ser den mängd fördelar som finns gentemot den handskrivna versionen. Vi har minskat antalet fält som ska skrivas in manuellt.

På dagens tillvägagångssätt fyller man i ca 20 fält men med applikationen minskar man riskerna för misstag som kan vara lätta att göra genom att de fält som användaren måste skriva in själv är få till antalet. Om man bortser från de fält som man läser in med streckkodsläsaren så är det bara 4 fält man behöver skriva in – kampanjpris, pris på tillbehör, ramnummer samt S-regnummer. En annan övergripande fördel är att man har en mall att följa. De garantikvitton som finns idag ger ingen vägledning. Det är lättare att lära sig applikationen gentemot det handskrivna kvittot. Tillvägagångssättet är väldigt enkelt att sätta sig in i samt lättare att komma ihåg.

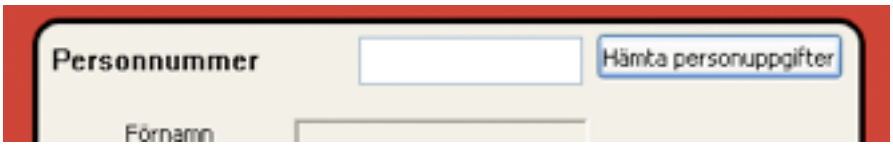
Vår målsättning var att skapa en prototyp som fungerar i Team Sportias system.

Vårt prototypprogram kan skriva ut garantikvitton och det kan hämta information från en Oracle- och en IBM-databas. Applikationen ska vara kompatibel med Team Sportias databaser men detta har vi inte kunnat testa. Men vi anser att den testmiljön vi skapat är så pass lik så att eventuella problem endast ska handla om fel namn på fält eller att någon dll-fil saknas.

7 Slutsatser och möjliga förbättringar

7.1 Möjliga förbättringar i applikationen

Gränssnittet fyller sin funktion väl. Men det finns små saker som går att förbättra. När man fyllt i första fältet (figur 17) och tryckt enter markeras inte nästa knapp inför nästkommande inmatning. Det här hade sparat någon sekund eftersom streckkodsläsaren automatisk sänder ett enterslag efter att inmatningen är klar.



Figur 17

Efter inmatning av person- och EAN-nummer skulle ett test av den inmatade informationen hjälpa användaren innan man försöker hämta informationen från databaserna. Nu kan det bli onödiga anrop ifall att personnumrets indatamask är felaktig. Nu är risken rätt liten eftersom de flesta inmatningar kommer att ske med streckkodsläsare.

En stor del som saknas är den omfattande testning som behövs innan verklig drift. Vilket förmodlingen lett till många förslag på förbättringar.

7.2 Val av programspråk

Vad vi kan se hade vi inte kunnat göra vår applikation bättre med ett annat programmeringsspråk. Visual C++ är ett så pass kraftfullt språk så att vi inte känt oss begränsade.

7.3 Databaskopplingar

Den teknik vi använde fungerade som vi hade tänkt. Eftersom det här är en av Microsofts senaste tekniker var den relativt lättanvänd. Av de tekniker som är tillgängliga är ADO-anslutningsträngar den som passat oss bäst. Vårt val av databasanslutning berodde mest på att den skulle se likadan ut oberoende av databas och att den skulle vara lätt att implementera och ändra i. Hade valet berott på prestanda istället hade det kanske funnits någon som var bättre. Det finns bl a en teknik som kallas OLE DB (Object Linking and Embedding for Databases) som skulle kunna vara snabbare eftersom den ligger något närmre

databaserna än ADO, ADO är nämligen ett lager på OLE DB. Vi tror att den prestandaminskning vi missat är minimal vilket inte påverkar vår applikation. Vi vann en hel del i användarvänlighet istället. En annan teknik som skulle fungerat att använda är DAO (Data Access Objects) men där fanns inte samma stöd för IBM- eller Oracles databaser.

7.4 Utskrift

Det utskriftsbibliotek, CDC, vi använde oss av hade de funktioner vi behövde. Det finns andra man kan använda men som är mer komplicerade när det gäller att skapa objekt. Ytterligare en sak som vore bra att lägga till är någon form av transaktionsnummer för att kunna spåra ett kvitto i databasen i de fall kunden kommer med ett garantiärende. Det här var något både vi och Team Sportia förbiså g.

7.5 Citrixanpassning

Programmet behöver anpassas till Citrix för att det ska fungera i Team Sportias butiker. Som vi tidigare nämnt behöver konfigurationsfilernas sökväg ändras.

7.6 Reflektioner

7.6.1 Övergång till målsystem

Övergången till Team Sportias system gick inte alls som vi tänkt. Vi kunde testa att skriva ut ett garantipapper precis som vi tänkt men våra databaskopplingar var värdelösa eftersom det inte gick att ansluta till databaserna över internet. Kopplingarna i sig fungerade på vårt testsystem och de hade fungerat hos Team Sportia också men eftersom vi inte kunde nå databaserna kunde vi inte komma vidare.

Vi hade kunnat upptäcka det här i vårt förarbete om vi hade hört med en systemutvecklare på Team Sportia. Men eftersom vi redan fått informationen av uppdragsgivaren att det skulle fungera var det den vi la upp vårt arbete efter.

7.6.2 Förbättringar av tidsplanering

När man lägger upp strategin för ett projekt innebär det också att man ska fördela den tid man fått på ett bra och realistiskt sätt. Genom att specificera de

delmoment som man ska ta sig igenom får man genast en greppbar överblick över vad som komma skall. Vi gjorde felet vid detta moment att vara alltför tidsoptimistiska samt antog att det praktiska i exjobbet skulle vara större än dokumentskrivningen. Då vi gång på gång fick göra om rapporten utefter handledarens önskemål sköts den praktiska delen upp längre. Det här berodde på att vi inte kunde sätta igång med programmeringen förrän vissa delmoment i rapporten var avklarade. Det måste till en bra tidsplan och en bra helhetssyn på vad rapporten ska innefatta innan den skapas. Annars är risken att vissa moment kommer på efterkälken och att resten av projektet måste stanna upp.

7.6.3 Resultatet

Tiden det tar att registrera ett garantikvitto är drygt 30 sekunder vilket är en minskning jämfört med dagens tillvägagångssätt. Gränssnittet är lättöverskådligt och hjälper den anställde med att utföra sin uppgift.

Vårt mål var att skapa ett prototypprogram som fungerar i Team Sportias befintliga Windowsmiljö. Vår prototyp fungerar i Windows 2000 och XP men databasanslutningarna fungerar inte i Team Sportias miljö. Målet var även att skapa ett garantikvitto vilket fungerar i prototypen.

Hade det inte varit för Citrix hade vi nått vårt mål. Men nu kom vi inte ändra fram. Applikationen förkortar registreringstiden och underlättar en hel del för personalen. Det är intressant att se hur mycket jobb man kan spara genom att utnyttja redan tillgänglig information från databaser. Felsäkerheten har ökat betydligt med tanke på de slarvfel som gärna slinker in annars vid t.ex. tidspress. Nu får man en mall att följa på ett helt annat sätt och man hämtar in personnummer och EAN-kod från streckkodsläsaren.

8 Referenser

1. Visual C++ .NET på 3 veckor
   Davis Chapman
   2002, Pagina

2. Print Previewing
   http://www.microsoft.com/msj/archive/S126A.aspx
   Jeff Prosise, MSDN
   (april 1996)

3. CDC-class
   http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vcmfc98/_mfc_cdc.asp
   MSDN
   (mars 2006)

4. Oracle 10g Express
   http://www.oracle.com/technology/products/database/xe/index.html
   Oracle
   (mars 2006)

5. IBM DB2 Express-C
   http://www-30.ibm.com/software/data/db2/udb/db2express/
   IBM
   (mars 2006)

6. Anslutningsträngar
   http://www.connectionstrings.com/
   ConnectionStrings.com
   (mars 2006)

7. Priser på utvecklingspaket
   http://www.dustin.se
   Dustin
   (mars 2006)

8. Citrix RevGuide
   http://support.citrix.com/servlet/KbServlet/download/9475-102-14686/RevGuide.pdf
   (maj 2006)
   Citrix

9. Hur man lägger bilder som bakgrund
http://www.functionx.com/visualc/applications/displaybitmap.htm
(maj 2006)
FunctionX

10. Forum
http://www.cplusplus.nu
(februari 2006)
CPlusPlus.nu

9 Bilagor

9.1 Bilaga A – Källkod

Här följer källkoden för applikationen. Om någon del av källkoden utelämnats står det under berörd rubrik. Längst ner följer även de textfilerna med anslutningssträngar för databaserna.

9.1.1 GuaranteeData

9.1.1.1 GuaranteeData.h

```
/*
Klassen har hand om garantidatan. Den skapar en cykel och en kund.
När GuaranteeData skapas skapas en tom Bike och Customer.
De uppdateras med data i updateCustomer respektive updateBike
Klassen sköter även butiksuppgifterna.
*/

#include "BikeDB.h"
#include "CustomerDB.h"
#include <stdio.h>

#include <fstream>
using std::ifstream;


class GuaranteeData
{

    public:
        GuaranteeData(){};


    Bike aBike;                 //En cykel
    Customer aCustomer;         //En kund
    CString cstrStoreName;      //Affärens namn
    CString cstrStorePhoneNr;   //Affärens telefonnummer
```

```
    CString cstrStoreAddress;   //Affärens adress
    CString cstrStoreCity;      //Staden affären ligger i
    CString cstrStoreNumber;    //Affärens nummer;

    Bike getBike();             //returnerar cykel
    Customer getCustomer();     //returnerar kund
    CString getStoreName();     //returnerar namnet p√• affären
    CString getStoreAddress();  //Returnerar adressen till affären
    CString getStoreCity();     //Returnerar stad och postnr
    CString getStorePhoneNr();  //Returnerar telefonnummer till affären
    CString getStoreNumber();   //Returnerar affärsnummer
    CString getDate();          //Returnerar dagens datum

    //Lägger till ram- och S-Reg-nummer till cykeln
    //Anropar allts√• cykelobjektets funktioner för att sätta nummerna.
    void addMoreBikeData(CString strFrameNr, CString strSRegNr);
    void setPrices(int iCampaignPrice,int iAccessoriesPrice); //Sätter
kampanjpris och tillbehörspris på cykel

    void updateStoreInformation(); //Uppdaterar affärens information
    void updateCustomer (CString strCustomerNr); //hämtar data från
kunddatabasen utifrån det kundnr som hämtas in och uppdaterar kunden som
skapats.
    void updateBike (CString strProductNr); //hämtar data från
artikeldatabasen med hjälp av produktnumret iProductNr

    CString Int2CString(int iC); //Konvertera int till CString

};
```

9.1.1.2 GuaranteeData.cpp

```
#include <iostream>
using namespace std;

#include "stdafx.h"
#include "GuaranteeData.h"

//returnerar cykel
Bike GuaranteeData::getBike()
{
    return aBike;
};

//returnerar kund
Customer GuaranteeData::getCustomer()
{
    return aCustomer;
};

//returnerar namnet på affären
```

```
CString GuaranteeData::getStoreName()
{
    return cstrStoreName;
};

//Returnerar adressen till affären
CString GuaranteeData::getStoreAddress()
{
    return cstrStoreAddress;
};

//Returnerar stad och postnr för affären
CString GuaranteeData::getStoreCity()
{
    return cstrStoreCity;
};

//Returnerar telefonnummer till affären
CString GuaranteeData::getStorePhoneNr()
{
    return cstrStorePhoneNr;
};

//Returnerar affärsnummer
CString GuaranteeData::getStoreNumber()
{
    return cstrStoreNumber;
};

//Hämta dagen datum
CString GuaranteeData::getDate()
{
    CString strDay;
    CTime ct = CTime::GetCurrentTime();
    strDay = ct.Format("%Y-%m-%d");

    return strDay;
};

//Sätter priset på cykeln
void GuaranteeData::setPrices(int iCampaignPrice,int iAccessoriesPrice)
{
    aBike.setCampaignPrice(iCampaignPrice);
    aBike.setAccessoriesPrice(iAccessoriesPrice);
};

//Lägger till ram- och S-Reg-nummer till cykeln
//Anropar alltså cykelobjektets funktioner för att sätta nummerna.
void GuaranteeData::addMoreBikeData(CString strFrameNr, CString
strSRegNr)
{
    aBike.setFrameNr(strFrameNr);
```

```
    aBike.setSRegNr(strSRegNr);
};

//Hämtar informationen från storeinformation.txt i configkatalogen
void GuaranteeData::updateStoreInformation()
{

    ifstream fin;
    fin.open("config/storeinformation.txt");
    string strTemp;

    getline(fin,strTemp);
    cstrStoreName = strTemp.c_str();
    getline(fin,strTemp);
    cstrStorePhoneNr = strTemp.c_str();
    getline(fin,strTemp);
    cstrStoreAddress = strTemp.c_str();
    getline(fin,strTemp);
    cstrStoreCity = strTemp.c_str();
    getline(fin,strTemp);
    cstrStoreNumber = strTemp.c_str();


    fin.close();

};
//hämtar data från kunddatabasen utifrån det kundnr som hämtas in och
uppdaterar kunden som skapats.
void GuaranteeData::updateCustomer (CString strCustomerNr)
{

    CustomerDB cDB;
    aCustomer = cDB.getCustomer(strCustomerNr);

};

 //hämtar data från artikeldatabasen med hjälp av produktnumret
iProductNr. Sedan uppdateras cykeln som skapats
void GuaranteeData::updateBike (CString strProductNr)
{

    //Sätt namnen
    BikeDB bDB;
    aBike = bDB.getBike(strProductNr);


};

//Konverterar från integer till CString
CString GuaranteeData::Int2CString(int iC)
{
CString cstrTemp;
```

```
            cstrTemp.Format(_T("%d"),iC);
    return cstrTemp;
};
```

9.1.2 Customer

9.1.2.1 Customer.h

```
/*
Klassen kund skapar en kund med namn och adressuppgifter.
*/

#include <iostream>
#include <cstdlib>
#include <string>
#include <stdio.h>


using namespace std;

class Customer
{
    public:

        Customer(){};

    CString strFirstName;   //Förnamn
    CString strLastName;        //Efternamn
    CString strAddress;     //Adress
    int iZipCode;           //Postnr
    CString strPhoneNr;     //Telefonnummer
    CString strCity;
    CString strCivicRegistrationNumber; //Personnummer

    CString getFirstName ();                //Returnerar förnamn
    CString getLastName ();             //Returnerar efternamn
    CString getAddress ();              //Returnerar adress
    int getZipCode ();                  //Returnerar postnr
    CString getPhoneNr ();              //Returnerar telefonnr
    CString getCity();                  //Returnerar stad
    CString getCivicRegistrationNumber();   //Returnerar personnummer

    void setFirstName (CString cstrFirstName);  //Sätter förnamn
    void setLastName (CString cstrLastName);    // Sätter efternamn
    void setAddress (CString cstrAddress);      // Sätter adress
    void setZipCode (int _iZipCode);            // Sätter postnr
    void setPhoneNr (CString cstrPhoneNr);      // Sätter telefonnr
    void setCity    (CString cstrCity);         //Sätter stad
    void setCivicRegistrationNumber(CString cstrCivicRegistrationNumber);
//Sätter personnummer

    void print();
};
```

9.1.2.2 Customer.cpp

```
#include "stdafx.h"
#include "customer.h"
//Returnerar förnamn
CString Customer::getFirstName ()
{
    return strFirstName;
};
//Returnerar efternamn
CString Customer::getLastName ()
{
    return strLastName;
};
//Returnerar adress
CString Customer::getAddress ()
{
    return strAddress;
};
//Returnerar postnr
int Customer::getZipCode ()
{
    return iZipCode;
};

//Returnerar telefonnr
CString Customer::getPhoneNr ()
{
    return strPhoneNr;
};


//Returnerar stad
CString Customer::getCity ()
{
    return strCity;
};

//Returnerar personnummer
CString Customer::getCivicRegistrationNumber()
{
    return strCivicRegistrationNumber;
};

//Sätter förnamn
void Customer::setFirstName (CString cstrFirstName)
{
    strFirstName = cstrFirstName;
};
// Sätter efternamn
void Customer::setLastName (CString cstrLastName)
{
    strLastName = cstrLastName;
```

```
};
// Sätter adress
void Customer::setAddress (CString cstrAddress)
{
    strAddress = cstrAddress;
};
// Sätter postnr
void Customer::setZipCode (int _iZipCode)
{
    iZipCode = _iZipCode;
};
// Sätter telefonnr
void Customer::setPhoneNr (CString cstrPhoneNr)
{
    strPhoneNr = cstrPhoneNr;
};
//Sätter personnummer
void Customer::setCity(CString cstrCity)
{
    strCity = cstrCity;
};

//Sätter personnummer
void Customer::setCivicRegistrationNumber(CString
cstrCivicRegistrationNumber)
{
    strCivicRegistrationNumber = cstrCivicRegistrationNumber;
};

//Skriver ut
void Customer::print()
{
    cout << "----Persondata----\n";
    cout<<"Förnamn: " << strFirstName << "\n";
    cout<<"Efternamn: " << strLastName << "\n";
};
```

9.1.3 Bike

9.1.3.1 Bike.h

```
 /*
Klassen har hand om en cykels egenskapar såsom märke, leverantör samt
återförsäljarnummer och garantitext.
*/

#include <iostream>
#include <cstdlib>
#include <string>
#include <stdio.h>
```

```
using namespace std;


#include <fstream>
using std::ifstream;
class Bike
{
    public:

        Bike(){};

    CString getFrameNr();       //Returnera ramnummer
    CString getColor ();        //Returnera färg på cykel
    CString getProductName ();  //Returnera artikelnamn
    CString getBrand ();        //Returnera Märke
    CString getSupplier();      //Returnera Leverantör
    CString getConfiguration(); //Returnera Konfiguration
    int getSize();              //Returnera Storlek
    CString getEAN();           //Returnera EAN-kod
    int getBarCode();           //Returnera Streckkod
    int getPrice();             //Returnerar priset
    int getCampaignPrice();     //Returnerar kampanjpris
    int getAccessoriesPrice();  //Returnerar tillbehörspris
    CString getProductNr();     //Returnerar produktnumret
    CString getRetailNr();      //Returnerar återförsäljarnummer
    CString getGuaranteeText(); //Returnerar garantitexten
    CString getSRegNr();        //Returnerar S-Regnummer


    void setFrameNr(CString strC);      //Sätter ramnummer
    void setColor (CString strC);       // Sätter färg
    void setProductName (CString strP); //Sätter artikelnamn
    void setBrand (CString strC);       //Sätter Märke
    void setSupplier(CString strC);     //Sätter Leverantör
    void setConfiguration(CString strC);//Sätter Konfiguration
    void setSize(int iC);               //Sätter Storlek
    void setEAN(CString strC);          //Sätter EAN-kod
    void setBarCode(int iC);            //Sätter Streckkod
    void setPrice(int iC);              //Sätter pris
    void setCampaignPrice(int iC);      //Sätter kampanjpris
    void setAccessoriesPrice(int iC);   //Sätter tillbehörspris
    void setProductNr(CString strP);    //Sätter produktnr
    void setRetailNr();                 //Sätter återförsäljarnummer
beroende på återförsäljare. Måste anropas efter setBrand
    void setGuaranteeInformation();     //Sätter garantitexten beroende
på återförsäljare
    void setSRegNr(CString strC);       //Sätter S-Regnummer

    private:
    CString strColour;      //Färg på cykel
    CString strProductName; //Artikelnamn
    CString strBrand;       //Märke
```

```
    CString strSupplier;    //Leverantör
    CString strConfiguration;//Konfiguration
    int iSize;              //Storlek
    CString strEAN;         //EAN-kod
    int iBarCode;           //Streckkod
    CString strFrameNr;
    int iPrice;             //Pris
    int iCampaignPrice;     //Kampanjpris
    int iAccessoriesPrice;  //Pris på tillbehör
    CString strProductNr;   //Produktnr
    CString cstrRetailNr;   //√Öterförsäljarnummer
    CString cstrGuaranteeText;  //Garantitext, skiljer sig åt beroende på
återförsäljare
    CString strSRegNr;          //S-Regnummer
    };
```

9.1.3.2 Bike.cpp

```
#include "stdafx.h"
#include "bike.h"

//Returnera  färg på cykel
CString Bike::getFrameNr ()
{
    return strFrameNr;
};

//Returnera  färg på cykel
CString Bike::getColor ()
{
    return strColour;
};

//Returnera  namn på produkt exempelvis Citybike.
CString Bike::getProductName ()
{
    return strProductName;
};

//Returnera  Märke
CString Bike::getBrand()
{
    return strBrand;
};

//Returnera  Leverantör
CString Bike::getSupplier()
{
    return strSupplier;
};

//Returnera  Konfiguration
```

```
CString Bike::getConfiguration()
{
    return strConfiguration;
};

//Returnera  Storlek
int Bike::getSize()
{
    return iSize;
};

//Returnera  EAN-kod
CString Bike::getEAN()
{
    return strEAN;
};

//Returnera  Streckkod
int Bike::getBarCode()
{
    return iBarCode;
};

//Returnera  priset
int Bike::getPrice()
{
    return iPrice;
};

//Returnera  kampanjpriset
int Bike::getCampaignPrice()
{
    return iCampaignPrice;
};

//Returnera  priset på tillbehör
int Bike::getAccessoriesPrice()
{
    return iAccessoriesPrice;
};

//Returnera  priset
CString Bike::getProductNr()
{
    return strProductNr;
};

//Returnerar återförsäljarnummer
CString Bike::getRetailNr()
{
    return cstrRetailNr;
};
```

```
//Returnerar S-Regnummer
CString Bike::getGuaranteeText()
{
    return cstrGuaranteeText;
};

//Returnerar S-Regnummer
CString Bike::getSRegNr()
{
    return strSRegNr;
};



// Sätter ramnummer
void Bike::setFrameNr (CString strC)
{
    strFrameNr = strC;
};

// Sätter färg
void Bike::setColor (CString strC)
{
    strColour = strC;
};

// Sätter produktnamn
void Bike::setProductName (CString strP)
{
    strProductName = strP;
};

//Sätter Märke
void Bike::setBrand (CString strC)
{
    strBrand = strC;
};

//Sätter Leverantör
void Bike::setSupplier(CString strC)
{
    strSupplier = strC;
    setRetailNr(); //Tar fram återförsäljarnr för cykel
    setGuaranteeInformation(); //Tar fram garantitext för återförsäljare
};

//Sätter Konfiguration
void Bike::setConfiguration(CString strC)
{
    strConfiguration = strC;
};
```

```
//Sätter Storlek
void Bike::setSize(int iC)
{
    iSize = iC;
};

//Sätter EAN-kod
void Bike::setEAN(CString strC)
{
    strEAN = strC;
};

//Sätter Streckkod
void Bike::setBarCode(int iC)
{
    iBarCode = iC;
};

//Sätter pris
void Bike::setPrice(int iC)
{
    iPrice = iC;
};

//Sätter kampanjpris
void Bike::setCampaignPrice(int iC)
{
    int test = 0;
    iCampaignPrice = iC;
};

//Sätter pris på tillbehör
void Bike::setAccessoriesPrice(int iC)
{
    int temp = 0;
    iAccessoriesPrice = iC;
    int iTemp = 3;
};

// Sätter produktnamn
void Bike::setProductNr (CString strP)
{
    strProductNr = strP;
};

//Returnerar återförsäljarnummer baserat på vad det är för märke på
cykeln
//Om det t.ex. är en cykel av märket Team så hämtas textfilen
retailerTeam.txt in.
void Bike::setRetailNr()
{
```

```
    CString strRetailerFile = _T("config/retailers/");
    strRetailerFile += strSupplier;
    strRetailerFile += _T("retailer.txt");


    ifstream fin;
    fin.open(strRetailerFile);
    string strRetailNr;
    getline(fin,strRetailNr);

    cstrRetailNr = strRetailNr.c_str();
    fin.close();

};

void Bike::setGuaranteeInformation()
{
    CString strRetailerFile = _T("config/information/");
    strRetailerFile += strSupplier;
    strRetailerFile += _T("information.txt");

    cstrGuaranteeText = _T("");

    ifstream fin;
    fin.open(strRetailerFile);
    string strGuaranteeText;
    while (getline(fin,strGuaranteeText))
    {
    cstrGuaranteeText += strGuaranteeText.c_str();
    cstrGuaranteeText +=  + _T("\n");
    }
    fin.close();

};

// Sätter S-Regnummer
void Bike::setSRegNr (CString strC)
{
    strSRegNr = strC;
};
```

9.1.4 PrintGuaranteeData

9.1.4.1 PrintGuaranteeData.h

```
/*
 Skriver ut garantidata med hjälp av den information som finns i
GuaranteeData
 */


#include <iostream>
#include <cstdlib>
```

```
#include <string>
#include <stdio.h>
using namespace std;

class PrintGuaranteeData
{
    public:

        PrintGuaranteeData(){};

    void printData(GuaranteeData gd);

    //Konvertera från int till CString
    CString Int2CString(int iC);
};
```

9.1.4.2 PrintGuaranteeData.cpp

```
/*
Skriver ut garantipapper
 */
#include <windows.h>

#include <stdio.h>

#include "stdafx.h"
#include "GuaranteeData.h"
#include "PrintGuaranteeData.h"


//Skriver ut data
void PrintGuaranteeData::printData(GuaranteeData gd)
{
    //Hämta in värden till CString:s och konvertera eventuella int till
CString.
            CString cstrFirstName = gd.getCustomer().getFirstName();
            CString cstrLastName = gd.getCustomer().getLastName();
            CString cstrPhoneNr = gd.getCustomer().getPhoneNr();
            CString cstrAddress = gd.getCustomer().getAddress();
            CString cstrZipCode =
Int2CString(gd.getCustomer().getZipCode());
            CString cstrCity = gd.getCustomer().getCity();

            CString cstrProductName = gd.getBike().getProductName();
            CString cstrSupplier = gd.getBike().getSupplier();
            CString cstrBrand = gd.getBike().getBrand();
            CString cstrPrice = Int2CString(gd.getBike().getPrice());
            CString cstrRetailerNr = gd.getBike().getRetailNr();
            CString cstrFrameNr = gd.getBike().getFrameNr();
            CString cstrSRegNr = gd.getBike().getSRegNr();
            CString cstrCampaignPrice =
Int2CString(gd.getBike().getCampaignPrice());
            CString cstrAccessoriesPrice =
Int2CString(gd.getBike().getAccessoriesPrice());
```

```
            CString cstrTotalPrice;
            //Räkna samman pris. Om kampanjpris är tomt ska ordinariepris
användas
            if (gd.getBike().getCampaignPrice() == 0)
            {
                cstrTotalPrice = Int2CString(gd.getBike().getPrice() +
gd.getBike().getAccessoriesPrice());
            }
            else
            {
                cstrTotalPrice =
Int2CString(gd.getBike().getCampaignPrice() +
gd.getBike().getAccessoriesPrice());
            }

            CString cstrDate = gd.getDate();
            CString cstrStoreName = gd.getStoreName();
            CString cstrStoreAddress = gd.getStoreAddress();
            CString cstrStoreNumber = gd.getStoreNumber();
            CString cstrStorePhoneNr = gd.getStorePhoneNr();

            CString cstrGuaranteeText = gd.getBike().getGuaranteeText();
```

```
    BOOL bContinue = TRUE;
    int nPageCount = 1;

    CDC dc;
    //Välj standardskrivare


    CPrintDialog dlg (FALSE);
    dlg.GetDefaults ();
    dc.Attach (dlg.GetPrinterDC ());

    DOCINFO di;
    ::ZeroMemory (&di, sizeof (DOCINFO));
    di.cbSize = sizeof (DOCINFO);
    di.lpszDocName = _T("Garantikvitto");



        CFont font;
```

```
font.CreateFont(
                70,
                0,
                0,
                0,
                FW_NORMAL,
                FALSE,
                FALSE,
                0,
                ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
                DEFAULT_PITCH | FF_SWISS,
                _T("Arial"));

CFont fontBold;
fontBold.CreateFont(
                70,
                0,
                0,
                0,
                FW_BOLD,
                FALSE,
                FALSE,
                0,
                ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
                DEFAULT_PITCH | FF_SWISS,
                _T("Arial"));
dc.SelectObject(&font);

CFont fontLabel;
fontLabel.CreateFont(
                100,
                0,
                0,
                0,
                FW_BOLD,
                FALSE,
                FALSE,
                0,
                ANSI_CHARSET,
                OUT_DEFAULT_PRECIS,
                CLIP_DEFAULT_PRECIS,
                DEFAULT_QUALITY,
                DEFAULT_PITCH | FF_SWISS,
                _T("Arial"));

CFont fontSmall;
```

```
        fontSmall.CreateFont(
                        50,
                        0,
                        0,
                        0,
                        FW_NORMAL,
                        FALSE,
                        FALSE,
                        0,
                        ANSI_CHARSET,
                        OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS,
                        DEFAULT_QUALITY,
                        DEFAULT_PITCH | FF_SWISS,
                        _T("Arial"));
        dc.SelectObject(&font);



    if (dc.StartDoc (&di) > 0)
    {


        for (int i=1; i<=nPageCount && bContinue; i++)
        {


    int iYValue = 80; //Värdet som texten ska förflyttas i y-led
    int tabValue_2 = 600; //Värdet som texten ska förflyttas vid tab 2
    int tabValue_1 = 150; //Värdet som texten ska förflyttas vid tab 1
    int firstLabel = 380; //Värdet i y led pÅ första labeln


    int rectHeight = 2400; //Värdet för höjden pÅ rektanglarna
    int rectWidth = 2080; //Värdet för höjden pÅ rektanglarna

    int iPartValue = 200; //Styckets placering i Y-led
    int iMultipleValue = 0;
    int iY; //Den variabel som används för att positionera textrutan.
Räknas ut var gÅng man ska flytta en ruta

            //Starta utskrift av sida
            dc.StartPage();
                dc.Rectangle(110, 280, 110 + rectWidth,110 + rectHeight);


            //Hämta in och placera Team Sportia-logo
            int imageWidth = 600;
            int imageHeight = 115;
            HBITMAP hBitmap =
(HBITMAP)::LoadImage(AfxGetInstanceHandle(),_T("config/teamsportia.bmp"),
IMAGE_BITMAP,0,0,LR_LOADFROMFILE | LR_CREATEDIBSECTION);
```

```
            CBitmap bmpTeamLogo;

            bmpTeamLogo.Attach(hBitmap);

            CDC memDCTeam;
            memDCTeam.CreateCompatibleDC(&dc);
            CBitmap *bmpPrevious = memDCTeam.SelectObject(&bmpTeamLogo);
            dc.StretchBlt(110 + rectWidth/2 -imageWidth/2, 120,
imageWidth, imageHeight, &memDCTeam,0, 0,168,32, SRCCOPY);
            dc.SelectObject(bmpPrevious);


                dc.SelectObject(&fontLabel);
                dc.TextOut(120,300,_T("Garantikvitto"));
                dc.SelectObject(&font);

                // här börjar utskriften av kunden
                int startPointY = firstLabel*1;

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold); //Välj fet font
                dc.TextOut(tabValue_1,iY,_T("fgare:"));
                dc.SelectObject(&font); //Välj vanlig font
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrFirstName += _T("
") + cstrLastName);

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Personnummer:"));
                dc.SelectObject(&font);

    dc.TextOut(tabValue_1+tabValue_2,iY,gd.getCustomer().getCivicRegistra
tionNumber());

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Adress:"));
                dc.SelectObject(&font);
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrAddress);

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrZipCode += _T("
") + cstrCity);

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Telefon:"));
```

```
dc.SelectObject(&font);
dc.TextOut(tabValue_1+tabValue_2,iY,cstrPhoneNr);


// här börjar utskriften av produkten
startPointY = iY + iPartValue;

iMultipleValue = 0;
iY = startPointY + iYValue*iMultipleValue;
dc.SelectObject(&fontBold);
dc.TextOut(tabValue_1,iY,_T("Märke:"));
dc.SelectObject(&font);
dc.TextOut(tabValue_1+tabValue_2,iY,cstrProductName);


iMultipleValue++;
iY = startPointY + iYValue*iMultipleValue;
dc.SelectObject(&fontBold);
dc.TextOut(tabValue_1,iY,_T("Tillverkare:"));
dc.SelectObject(&font);
dc.TextOut(tabValue_1+tabValue_2,iY,cstrSupplier);


iMultipleValue++;
iY = startPointY + iYValue*iMultipleValue;
dc.SelectObject(&fontBold);
dc.TextOut(tabValue_1,iY,_T("≈terförsäljarnr:"));
dc.SelectObject(&font);
dc.TextOut(tabValue_1+tabValue_2,iY,cstrRetailerNr);



iMultipleValue++;
iY = startPointY + iYValue*iMultipleValue;
dc.SelectObject(&fontBold);
dc.TextOut(tabValue_1,iY,_T("Ramnummer:"));
dc.SelectObject(&font);
dc.TextOut(tabValue_1+tabValue_2,iY,cstrFrameNr);

iMultipleValue++;
iY = startPointY + iYValue*iMultipleValue;
dc.SelectObject(&fontBold);
dc.TextOut(tabValue_1,iY,_T("S-Reg.Nummer:"));
dc.SelectObject(&font);
dc.TextOut(tabValue_1+tabValue_2,iY,cstrSRegNr);


// här börjar utskriften av priser
startPointY = iY + iPartValue;

iMultipleValue = 0;
iY = startPointY + iYValue*iMultipleValue;
```

```
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Pris:"));
                dc.SelectObject(&font);
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrPrice);


                if (gd.getBike().getCampaignPrice() != 0)
                {
                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Kampanjpris:"));
                dc.SelectObject(&font);
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrCampaignPrice);
                }

                if (gd.getBike().getAccessoriesPrice() != 0)
                {
                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Tillbehörspris:"));
                dc.SelectObject(&font);

    dc.TextOut(tabValue_1+tabValue_2,iY,cstrAccessoriesPrice);
                }

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue+10;
                dc.MoveTo(tabValue_1,iY-1);
                dc.LineTo(tabValue_1 + tabValue_2 + 200,iY-10);
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Summa:"));
                dc.SelectObject(&font);
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrTotalPrice);



                // här börjar utskriften av adressuppgifter för butik
                startPointY = iY + iPartValue;

                iMultipleValue = 0;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
                dc.TextOut(tabValue_1,iY,_T("Försäljningsdatum:"));
                dc.SelectObject(&font);
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrDate);

                startPointY = iY + iPartValue;
                iMultipleValue = 0;
                iY = startPointY + iYValue*iMultipleValue;
                dc.SelectObject(&fontBold);
```

```
                dc.TextOut(tabValue_1,iY,_T("Säljare:"));
                dc.SelectObject(&font);
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrStoreName);

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrStoreAddress);

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrStorePhoneNr);

                iMultipleValue++;
                iY = startPointY + iYValue*iMultipleValue;
                dc.TextOut(tabValue_1+tabValue_2,iY,cstrStoreNumber);


                //Skriv ut anmärkning
                dc.SelectObject(&fontSmall);
                dc.TextOut(130,110 + rectHeight + 10,_T("OvanstÅende
uppgifter registreras i S Reg och blir därmed även"));

                dc.TextOut(130,110 + rectHeight + 70,_T("tillgängliga för
polis, försäkringsbolag och cykelleverantör."));

                //Skriv ut garantitext
                CRect rect;
                rect.SetRect(110,110 + rectHeight + 150,rectWidth +
110,110 + rectHeight + 900);
                dc.DrawText(cstrGuaranteeText,rect,DT_LEFT |
DT_WORDBREAK);




            if (dc.EndPage () <= 0)
            {
                bContinue = FALSE;
            }
            font.DeleteObject();
        }
        if (bContinue)
        {
            dc.EndDoc ();
        }
        else
        {
            dc.AbortDoc ();
        }
    }

dc.DeleteDC();
```

```
 //MessageBox(NULL,_T("Slut"),NULL,NULL);

};

//Konverterar frÅn integer till CString
CString PrintGuaranteeData::Int2CString(int iC)
{
CString cstrTemp;
            cstrTemp.Format(_T("%d"),iC);
    return cstrTemp;
};
```

9.1.5 SaveGuaranteeData

9.1.5.1 SaveGuaranteeData.h

```
/*
Klassen SaveGuaranteeData kommunicerar med garantidatabasen. Eftersom
vi fortfarande inte vet vilken form av databas som det ska sparas det
hela i bÅde IBM och Oracle för nuvarande
Den här klassen används av CykelregistreringDlg

DÅ det ska vara lätt att ändra anslutningssträngen med lösenord
och liknande hämtas den frÅn
ORAguaranteedb.txt
och
IBMguaranteedb.txt

 */

#include <iostream>
#pragma once
#include <fstream>
using namespace std;


#include <stdio.h>
#pragma once
#define INITGUID
#import "C:\Program\Delade filer\System\ado\msado15.dll"
rename_namespace("ADOCG") rename("EOF", "EndOfFile")
using namespace ADOCG;
#include "icrsint.h"


class SaveGuaranteeData
{
    public:
    SaveGuaranteeData(){};

    //oraSave sparar garantidatan i en Oracle-databas
    void oraSave(GuaranteeData gd);
```

```
    //ibmSave sparar garantidatan i en IBM-databas
    void ibmSave(GuaranteeData gd);

    private:
    //Returnerar rätt anslutningssträng
    CString getConnectionStringOra();
    CString getConnectionStringIBM();

    CString Int2CString(int iC);


};
```

9.1.5.2 SaveGuaranteeData.cpp

```
#include "stdafx.h"
#include "GuaranteeData.h"
#include "SaveGuaranteeData.h"

void SaveGuaranteeData::oraSave(GuaranteeData gd)
{
 // Initialisera Recordset
   _RecordsetPtr m_ptrRs = NULL;


    //Skapa variabler för det som ska sparas
   //Eftersom SQL-strängen behöver CString mÅste alla int konverteras
   CString cstrProductNr = gd.getBike().getProductNr();
    CString cstrProductName = gd.getBike().getProductName();
    CString cstrSupplier = gd.getBike().getSupplier();
    CString cstrBrand = gd.getBike().getBrand();
    CString cstrPrice = Int2CString(gd.getBike().getPrice());
    CString cstrCampaignPrice =
Int2CString(gd.getBike().getCampaignPrice());
    CString cstrAccessoriesPrice =
Int2CString(gd.getBike().getAccessoriesPrice());

    CString cstrCivicRegistrationNumber =
gd.getCustomer().getCivicRegistrationNumber();
    CString cstrFirstName = gd.getCustomer().getFirstName();
    CString cstrLastName = gd.getCustomer().getLastName();
    CString cstrAddress = gd.getCustomer().getAddress();
    CString cstrPhoneNr = gd.getCustomer().getPhoneNr();
    CString cstrCity = gd.getCustomer().getCity();
    CString cstrZipCode = Int2CString(gd.getCustomer().getZipCode());

    gd.updateStoreInformation(); //Uppdatera affärens information innan
den ska skrivas ut
    CString cstrStoreNr = gd.getStoreNumber();
    CString cstrRetailNr = gd.getBike().getRetailNr();
    CString cstrDate = gd.getDate();
```

```
    CString cstrFrameNr = gd.getBike().getFrameNr();
    CString cstrSRegNr = gd.getBike().getSRegNr();


//SQL-satsen som hämtar data frÅn produktdatabasen
CString cstrSQL = _T("INSERT INTO guaranteedata
(personnummer,fnamn,enamn,telefonnr,adress,postnr,stad,productnr,brand,pr
oductname,supplier,price,campaignprice,accessoriesprice,storenr,retailnr,
datum,framenr,sregnr) VALUES (");
cstrSQL +=  cstrCivicRegistrationNumber + _T(",'") +
            cstrFirstName + _T("','") +
            cstrLastName + _T("','") +
            cstrPhoneNr + _T("','") +
            cstrAddress + _T("',") +
            cstrZipCode + _T(",'") +
            cstrCity + _T("','") +
            cstrProductNr + _T("','") +
            cstrBrand + _T("','") +
            cstrProductName + _T("','") +
            cstrSupplier + _T("',") +
            cstrPrice + _T(",") +
            cstrCampaignPrice + _T(",") +
            cstrAccessoriesPrice + _T(",'") +
            cstrStoreNr + _T("','") +
            cstrRetailNr + _T("','") +
            cstrDate + _T("','") +
            cstrFrameNr + _T("','")+
            cstrSRegNr + _T("')");

MessageBox(NULL,cstrSQL,_T("Oracle SQL-sats"),NULL);
CString m_strCmdText = cstrSQL;
CString m_strConnection = getConnectionStringOra();


    ::CoInitialize(NULL);
    try
    {


            _ConnectionPtr ptrConn;
            ptrConn.CreateInstance(__uuidof(Connection));
            ptrConn-
>Open((LPCTSTR)m_strConnection,L"",L"",adConnectUnspecified);

            ptrConn->Execute((LPCTSTR)cstrSQL,NULL,adCmdUnknown);

    }
    // Skriv ut eventuellt felmeddelande i en MessageBox
    catch(_com_error &e)
    {
        MessageBox(NULL,e.Description(),e.ErrorMessage(),NULL);
    }
```

```
};

//Sparar garantidatan i en ibm-databas
void SaveGuaranteeData::ibmSave(GuaranteeData gd)
{
 // Initialisera Recordset
   _RecordsetPtr m_ptrRs = NULL;


    //Skapa variabler för det som ska sparas
    //Eftersom SQL-strängen behöver CString mÅste alla int konverteras
    CString cstrProductNr = gd.getBike().getProductNr();
    CString cstrProductName = gd.getBike().getProductName();
    CString cstrSupplier = gd.getBike().getSupplier();
    CString cstrBrand = gd.getBike().getBrand();
    CString cstrPrice = Int2CString(gd.getBike().getPrice());
    CString cstrCampaignPrice =
Int2CString(gd.getBike().getCampaignPrice());
    CString cstrAccessoriesPrice =
Int2CString(gd.getBike().getAccessoriesPrice());

    CString cstrCivicRegistrationNumber =
gd.getCustomer().getCivicRegistrationNumber();
    CString cstrFirstName = gd.getCustomer().getFirstName();
    CString cstrLastName = gd.getCustomer().getLastName();
    CString cstrAddress = gd.getCustomer().getAddress();
    CString cstrPhoneNr = gd.getCustomer().getPhoneNr();
    CString cstrCity = gd.getCustomer().getCity();
    CString cstrZipCode = Int2CString(gd.getCustomer().getZipCode());

    gd.updateStoreInformation(); //Uppdatera affärens information innan
den ska skrivas ut
    CString cstrStoreNr = gd.getStoreNumber();
    CString cstrRetailNr = gd.getBike().getRetailNr();
    CString cstrDate = gd.getDate();
    CString cstrFrameNr = gd.getBike().getFrameNr();
    CString cstrSRegNr = gd.getBike().getSRegNr();


//SQL-satsen som hämtar data frÅn produktdatabasen
CString cstrSQL = _T("INSERT INTO guaranteedata
(personnummer,fnamn,enamn,telefonnr,adress,postnr,stad,productnr,brand,pr
oductname,supplier,price,campaignprice,accessoriesprice,storenr,retailnr,
datum,framenr,sregnr) VALUES (");
cstrSQL +=  cstrCivicRegistrationNumber + _T(",'") +
            cstrFirstName + _T("','") +
            cstrLastName + _T("','") +
            cstrPhoneNr + _T("','") +
            cstrAddress + _T("',") +
            cstrZipCode + _T(",'") +
            cstrCity + _T("','") +
```

```
            cstrProductNr + _T("','") +
            cstrBrand + _T("','") +
            cstrProductName + _T("','") +
            cstrSupplier + _T("',") +
            cstrPrice + _T(",") +
            cstrCampaignPrice + _T(",") +
            cstrAccessoriesPrice + _T(",'") +
            cstrStoreNr + _T("','") +
            cstrRetailNr + _T("','") +
            cstrDate + _T("','") +
            cstrFrameNr + _T("','")+
            cstrSRegNr + _T("')");

MessageBox(NULL,cstrSQL,_T("IBM SQL-sats"),NULL);
CString m_strCmdText = cstrSQL;
CString m_strConnection = getConnectionStringIBM();


    ::CoInitialize(NULL);
    try
    {


            _ConnectionPtr ptrConn;
            ptrConn.CreateInstance(__uuidof(Connection));
            ptrConn-
>Open((LPCTSTR)m_strConnection,L"",L"",adConnectUnspecified);

            ptrConn->Execute((LPCTSTR)cstrSQL,NULL,adCmdUnknown);

    }
    // Skriv ut eventuellt felmeddelande i en MessageBox
    catch(_com_error &e)
    {
        MessageBox(NULL,e.Description(),e.ErrorMessage(),NULL);
    }

};


//Hämta anslutningssträng för Oracle
CString SaveGuaranteeData::getConnectionStringOra()
{

    ifstream fin;
    fin.open("config/connectionstrings/oraguaranteedb.txt");
    string strConnectionString;
    getline(fin,strConnectionString);
    fin.close();

    //Konvertera till CString för att kunna returnera
    CString returnValue;
```

```
    returnValue = strConnectionString.c_str();
    return returnValue;
};

//Hämta anslutningssträng för IBM-anslutningen
CString SaveGuaranteeData::getConnectionStringIBM()
{

    ifstream fin;

    fin.open("config/connectionstrings/ibmguaranteedb.txt");
    string strConnectionString;
    getline(fin,strConnectionString);
    fin.close();

    //Konvertera till CString för att kunna returnera
    CString returnValue;
    returnValue = strConnectionString.c_str();
    return returnValue;
};

//Konverterar frÅn integer till CString
CString SaveGuaranteeData::Int2CString(int iC)
{
CString cstrTemp;
            cstrTemp.Format(_T("%d"),iC);
    return cstrTemp;
};
```

9.1.6 BikeDB

9.1.6.1 BikeDB.h

```
/*
Klassen BikeDB kommunicerar med produktdatabasen.
Då det ska vara lätt att ändra anslutningssträngen med lösenord och
liknande hämtas den från productdb.txt
 */

#include <stdio.h>
#include "Bike.h"


#include <fstream>
using std::ifstream;

#pragma once
#define INITGUID
#import "C:\Program\Delade filer\System\ado\msado15.dll"
rename_namespace("ADOCG") rename("EOF", "EndOfFile")
using namespace ADOCG;
#include "icrsint.h"
```

```
class BikeDB
{
    public:
    BikeDB(){};

    //getBike returnerar en Bike efter att ha skapat den utifrån den data
som finns i Databasen.
    Bike getBike(CString strProductNr);


};
```

9.1.6.2 BikeDB.cpp

```
#include "stdafx.h"
#include "BikeDB.h"


Bike BikeDB::getBike(CString strEANnr)
{
    //Hämta in anslutningssträng
    ifstream fin;
    fin.open("config/connectionstrings/productdb.txt");
    string strConnectionString;
    getline(fin,strConnectionString);
    fin.close();


 // Initialisera Recordset
   _RecordsetPtr m_ptrRs = NULL;

    //Skapa variabler
    CString cstrColour;
    CString cstrProductName;
    CString cstrSupplier;
    CString cstrBrand;
    int iPrice = 0;

//SQL-satsen som hämtar data frÅn produktdatabasen
CString cstrSQL = _T("SELECT * FROM produkt WHERE streckean = '");
cstrSQL += strEANnr;
cstrSQL += "'";
CString m_strCmdText = cstrSQL;

//Konvertera string till CString för anslutningssträngen
    CString m_strConnection;
    m_strConnection = strConnectionString.c_str();
```

```
    ::CoInitialize(NULL);
    try
    {

            _ConnectionPtr ptrConn;
            ptrConn.CreateInstance(__uuidof(Connection));
            ptrConn-
>Open((LPCTSTR)m_strConnection,L"",L"",adConnectUnspecified);

            _CommandPtr ptrCmd;
            ptrCmd.CreateInstance(__uuidof(Command));

            ptrCmd->ActiveConnection = ptrConn;
            ptrCmd->CommandText = (LPCTSTR)m_strCmdText;

            _RecordsetPtr m_ptrRs;

            m_ptrRs.CreateInstance(__uuidof(Recordset));
            m_ptrRs->PutRefSource(ptrCmd);


            //Skapa varianten NULL
            _variant_t vNull;
            vNull.vt = VT_ERROR;
            vNull.scode = DISP_E_PARAMNOTFOUND;

            m_ptrRs-
>Open(vNull,vNull,adOpenDynamic,adLockOptimistic,adCmdUnknown);


         //Om det finns en rad med data ska den hämtas in.
        if(!m_ptrRs->EndOfFile)
        {
            cstrProductName = m_ptrRs-
>GetCollect(_variant_t("ARTIKELNAMN"));
            cstrColour = m_ptrRs->GetCollect(_variant_t("FƒRG"));
            cstrBrand = m_ptrRs->GetCollect(_variant_t("BRAND"));
            cstrSupplier = m_ptrRs->GetCollect(_variant_t("LEVERANT÷R"));
            iPrice = m_ptrRs->GetCollect(_variant_t("PRIS"));
        }
        else
        {
           MessageBox(NULL,_T("Kan inte hitta den angivna
produkten."),NULL,NULL);
        }

        m_ptrRs->Close();
        ptrConn->Close();

    }
    // Skriv ut eventuellt felmeddelande i en MessageBox
    catch(_com_error &e)
```

```
    {
        MessageBox(NULL,e.Description(),e.ErrorMessage(),NULL);
    }

    Bike aBike;

    aBike.setColor(cstrColour);
    aBike.setProductName(cstrProductName);
    aBike.setPrice(iPrice);
    aBike.setBrand(cstrBrand);
    aBike.setSupplier(cstrSupplier);
    aBike.setProductNr(strEANnr);
    return aBike;
};
```

9.1.7 CustomerDB

9.1.7.1 CustomerDB.h

```
/*
Klassen CustomerDB kommunicerar med kunddatabasen och returnerar en
Customer när den hämtat datan.
Då det ska vara lätt att ändra anslutningssträngen med lösenord och
liknande hämtas den från productdb.txt
Den här klassen blev till för att göra GuaranteeData mer lättläst
 */

#include "Customer.h"
#include <stdio.h>

#include <fstream>
using std::ifstream;

#pragma once
#define INITGUID
#import "C:\Program\Delade filer\System\ado\msado15.dll"
rename_namespace("ADOCG") rename("EOF", "EndOfFile")
using namespace ADOCG;
#include "icrsint.h"



class CustomerDB
{
    public:
    CustomerDB(){};

    //getCustomer returnerar en Customer efter att ha skapat den utifrån
den data som finns i Databasen.
    Customer getCustomer(CString strCivicRegistrationNumber);
```

```
    //SQLWCHAR* convertToStar(char* aString);


};
```

9.1.7.2 CustomerDB.cpp

```
#include "stdafx.h"
#include "CustomerDB.h"

Customer CustomerDB::getCustomer(CString strCivicRegistrationNumber)
{
    //Hämta in anslutningssträng
    ifstream fin;
    fin.open("config/connectionstrings/customerdb.txt");
    string strConnectionString;
    getline(fin,strConnectionString);
    fin.close();


 // Initialisera Recordsetet
   _RecordsetPtr m_ptrRs = NULL;

    //Skapa variabler som data ska hämtas in i. De används sedan för att
spara datan till personobjektet.
    CString cstrFirstName;
    CString cstrLastName;
    CString cstrAddress;
    CString cstrTelephone;
    CString cstrCity;
    int iZipCode = 0;


    //SQL-strängen skapas
    CString cstrSQL = _T("SELECT * FROM kunder WHERE PERSONNUMMER = '");
    cstrSQL += strCivicRegistrationNumber;
    cstrSQL += "'";
    CString m_strCmdText = cstrSQL;

    //Konvertera string till CString för anslutningssträngen
    CString m_strConnection;
    m_strConnection = strConnectionString.c_str();


    ::CoInitialize(NULL);
    try
    {
        _ConnectionPtr ptrConn;
        ptrConn.CreateInstance(__uuidof(Connection));
```

```
        ptrConn-
>Open((LPCTSTR)m_strConnection,L"",L"",adConnectUnspecified);
        _CommandPtr ptrCmd;
        ptrCmd.CreateInstance(__uuidof(Command));

        ptrCmd->ActiveConnection = ptrConn;
        ptrCmd->CommandText = (LPCTSTR)m_strCmdText;

        _RecordsetPtr m_ptrRs;

        m_ptrRs.CreateInstance(__uuidof(Recordset));
        m_ptrRs->PutRefSource(ptrCmd);


//Skapa varianten NULL
        _variant_t vNull;
        vNull.vt = VT_ERROR;
        vNull.scode = DISP_E_PARAMNOTFOUND;

        m_ptrRs-
>Open(vNull,vNull,adOpenDynamic,adLockOptimistic,adCmdUnknown);


       if(!m_ptrRs->EndOfFile)
       {
       cstrFirstName = m_ptrRs->GetCollect(_variant_t("FORNAMN"));
       cstrLastName = m_ptrRs->GetCollect(_variant_t("EFTERNAMN"));
       cstrAddress = m_ptrRs->GetCollect(_variant_t("ADRESS"));
       cstrTelephone = m_ptrRs->GetCollect(_variant_t("HEMNR"));
       cstrCity = m_ptrRs->GetCollect(_variant_t("ORT"));
       iZipCode = m_ptrRs->GetCollect(_variant_t("POSTNR"));

       }
       else
       {
          MessageBox(NULL,_T("Det går inte att hitta någon person med det
här personnumret"),NULL,NULL);
       }
       //**//
      m_ptrRs->Close();
      ptrConn->Close();

    }
    // Any errors?
    catch(_com_error &e)
    {

        MessageBox(NULL,e.Description(),NULL,NULL);
    }

    Customer aCustomer;
```

```
    aCustomer.setFirstName(cstrFirstName);
    aCustomer.setLastName(cstrLastName);
    aCustomer.setAddress(cstrAddress);
    aCustomer.setPhoneNr(cstrTelephone);
    aCustomer.setCity(cstrCity);
    aCustomer.setZipCode(iZipCode);
    aCustomer.setCivicRegistrationNumber(strCivicRegistrationNumber);


    return aCustomer;
};
```

9.1.8 CykelregistreringDlg

Här visas bara det som kodats av oss. Den här filen innehåller mycket automatiskt genererad kod som skapades vid gränssnittsbyggandet.

```
#include "stdafx.h"
#include "Cykelregistrering.h"
#include "CykelregistreringDlg.h"
#include "GuaranteeData.h"
#include "PrintGuaranteeData.h"
#include "SaveGuaranteeData.h"

BOOL CCykelregistreringDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    //Skapa typsnitt
    CFont cfont;
    cfont.Detach();
    cfont.CreateFont(
                        80,
                        0,
                        0,
                        0,
                        FW_BOLD,
                        FALSE,
                        FALSE,
                        0,
                        ANSI_CHARSET,
                        OUT_DEFAULT_PRECIS,
                        CLIP_DEFAULT_PRECIS,
                        DEFAULT_QUALITY,
                        DEFAULT_PITCH | FF_SWISS,
                        _T("Arial"));

staticCivicRegistrationNr.SetFont(&cfont);
staticEANnr.SetFont(&cfont);

}
```

```
void CCykelregistreringDlg::OnBnClickedbtngetcustomer()
{

    txtEANnr.SetFocus();



UpdateData(TRUE);







    gd.updateCustomer(strCivicRegistrationNr);

    strFirstName = gd.getCustomer().getFirstName();
    strLastName = gd.getCustomer().getLastName();
    strAddress = gd.getCustomer().getAddress();
    strPhoneNr = gd.getCustomer().getPhoneNr();
    strCity = gd.getCustomer().getCity();
    iZipCode = gd.getCustomer().getZipCode();

    //Uppdatera formuläret frÅn de kopplade variablerna
    UpdateData(FALSE);
}

void CCykelregistreringDlg::OnBnClickedbtngetbike()
{
CWnd * myedit = GetDlgItem(IDC_txtCampaignPrice);
        myedit->SetFocus();

//Hämta in all data frÅn formuläret. Datan läggs in i de kopplade
variablerna
    UpdateData(TRUE);


    gd.updateBike(strProductNr);

    strBrand = gd.getBike().getBrand();
    strProductName = gd.getBike().getProductName();
    strSupplier = gd.getBike().getSupplier();
    iPrice = gd.getBike().getPrice();

    //Uppdatera formuläret frÅn de kopplade variablerna
    UpdateData(FALSE);


}

void CCykelregistreringDlg::OnBnClickedButton1()
{
```

```
    UpdateData(TRUE);

    //PrintGuaranteeData pgd;


    SaveGuaranteeData sgd;

    gd.addMoreBikeData(strFrameNr,strSRegNr);
    gd.setPrices(intCampaignPrice,intAccessoriesPrice);
    gd.updateStoreInformation();



    gd.getDate();

    //pgd.printData(gd); //Skriv ut
    sgd.oraSave(gd);  //Spara i Oracle
    sgd.ibmSave(gd); //Spara i IBM

    UpdateData(FALSE);




}

//Rensar alla fält
void CCykelregistreringDlg::OnBnClickedButton2()
{


    //btnGetCustomer

UpdateData(TRUE);



    strCivicRegistrationNr = _T("");

    strFirstName = _T("");
    strLastName = _T("");
    strAddress = _T("");
    strPhoneNr = _T("");
    strCity = _T("");
    iZipCode = 0;

    strProductNr = _T("");
    strBrand = _T("");
    strProductName = _T("");
    strSupplier = _T("");
    iPrice = 0;
```

```
    intCampaignPrice = 0;
    intAccessoriesPrice = 0;

    //Uppdatera formuläret frÅn de kopplade variablerna
    UpdateData(FALSE);

    TxtCivicRegistrationNumber.SetFocus();
    //strCivicRegistrationNr
}
```

9.1.9 Customerdb.txt

Det här är anslutningssträngen för kunddatabasen.

Dsn=IBMDB;uid=Daniel;dbalias=CUSTOMER

9.1.10 Productdb.txt

Det här är anslutningssträngen för produktdatabasen.

Dsn=OracleDB;uid=SYSTEM;dba=W;apa=T;exc=F;fen=T;qto=T;frc=10;fdl=10;lob=T;rst=T;btd=F;bam=IfAllSuccessful;num=NLS;dpm=F;mts=T;mdi=F;csr=F;fwc=F;fbs=64000;tlo=O;Password=lösenord

9.1.11 Övriga anslutningssträngar

Det finns två filer till för anslutningssträngar ifall att garantidatan ska sparas i andra databaser än produkt- eller kunddatabasen.

9.2 Bilaga B – Dokument för Team Sportias tekniska förutsättningar

Här nedan följer det dokument vi skapade med Team Sportias tekniska förutsättningar. Det här dokumentet är det vi skapade i början av exjobbet innan vi fick reda på att man inte kan ansluta till databaserna över internet utan via en Citrix-server.

Team Sportias tekniska förutsättningar 2006-02-22

Operativsystem: Windows 2000 Service Pack 4 eller Windows XP Service Pack 2

Prestanda: Datorerna är inte äldre än 2 år gamla.

Nätverk: Samtliga datorer är anslutna till varanda och mot internet. Möjlighet att ansluta till databaserna över internet finns.

Program: Vissa har Officepaketet installerat men annar skiljer det stort mellan olika datorer.

Streckkodsläsare: Den dator programmet ska köras på har läsare men de andra ska ha möjlighet att köra programmet ändå.

Drivrutiner: Ifall att drivrutinerna inte finns på datorerna vi ska köra programmet ifrån får vi installera dem.

9.3 Bilaga C – Installations- och konfigurationsanvisningar

Dokumentet beskriver hur man installerar applikationen. Det här beror på att vi inte skapat något program som installerar programmet.

1. Kopiera mappen ”Garantiregistrering” från CD-skivan till valfri plats på datorn.
2. Öppna den överkopierade mappen och öppna undermappen ”config”. Här finns alla konfigurationsfiler.

9.3.1 Ändra anslutningsträngarna

Nedan följer var man ändrar anslutningsträngarna för respektive databas. Filerna för anslutningssträngarna ligger i mappen ”connectionstrings”

9.3.1.1 Kunddatabasen
Anslutningsträngen heter ”customerdb.txt”.

9.3.1.2 Produktdatabasen
Anslutningsträngen heter ”productdb.txt”

9.3.1.3 Garantidatabasen för IBM
Anslutningsträngen heter ”ibmguaranteedata.txt”

9.3.1.4 Garantidatabasen för Oracle
Anslutningsträngen heter ”oraguaranteedata.txt”

9.3.2 Garantitext och information

Garantitext och annan information som ska stå på kvittot ligger i mappen ”information”. Eftersom den beror på tillverkare börjar filens namn med tillverkarnamn och avslutas med information. T.ex. benämns Team Sportias egna cyklar med Team som tillverkar och följaktligen blir namnet på filen ”Teaminformation.txt”. Har man flera olika tillverkare skapar man en fil för varje. Det finns även en fil som heter endast ”information.txt”. Det är i det fallet det inte finns någon garantitext för tillverkaren. Här lägger man en standardiserad garantitext.

9.3.3 Återförsäljarnummer

Varje tillverkare av cyklar har ett visst återförsäljarnummer som ska stå på kvittot. Numret för respektive tillverkare hittar man i mappen ”retailers”. På samma sätt som för garantitext och annan information börjar filernas namn på tillverkarnamn men här följs det av ”retailer”. En cykel från Team Sportias eget märke heter ”Teamretailer.txt”.

9.3.4 Butikens adress- och telefonuppgifter

Uppgifterna står i filen ”storeinformation.txt”. Applikationen hämtar in en rad i taget och lägger det i en specifik variabel. Därför är det viktigt att ordningen är rätt i textfilen.

Ordningen är följande:

1. Butiksnamn
2. Telefonnummer
3. Adress
4. Postnummer och postadress
5. Butiksnummer

9.4 Bilaga D – Installationspaket för att kunna köra applikationen

Det här är de installationspaket som behövs för att få applikationen att fungera.

Microsoft Visual C++ 2005 Redistributable Package. Det går inte att installera paketet om man saknar Windows Installer 3.1. I Windows XP Service Pack 2 finns det redan förinstallerat.

9.5 Bilage E – CD-skivans mappträd

- Garantiregistrering
  - Cykelregistrering.exe
  - config
    - logo.gif
    - connectionstrings
      - customerdb.txt
      - oraguaranteedb.txt
      - productdb.txt
      - ibmguaranteedb.txt
    - information
      - Teaminformation.txt
      - information.txt
    - retailers
      - retailer.txt
      - Teamretailer.txt
    - storeinformation.txt

9.6 Bilaga F – Kravspecifikation

Kravspecifikation

Innehållsförteckning

1 Introduktion

1.1 Beskrivning av projektet

Projektet är ett exjobb för att ta fram en applikation som förenklar försäljningen av cyklar. Genom att använda befintliga databaser för kunder samt produkter skall ett garantikvitto kunna skrivas ut.

1.2 Målsättningar för projektet

Målet är att erhålla Team Sportia med en installerbar CD som innehåller vår applikation och de systemfiler som kommer att behövas för att köra det.

1.3 Beskrivningar av de inblandade parterna

Team Sportia är ett företag som säljer sportartiklar. I sortimentet ingår bl a cyklar. Anders Bergh och Daniel Nord är studenter vid Lunds Tekniska Högskola och ska utveckla en prototyp garantiregistrering av cyklar.

2 Systemkrav

2.1 Funktionella krav

2.1.1 Gränssnitt

2.1.1.1 Layouten skall ha ett lättförståligt upplägg

2.1.1.2 Applikationen skall max ha 4 inmatningsfällt

2.1.1.3 Minimalt antal knapptryckningar skall behövas vid användning

2.1.1.4 Färgvalet för applikationen skall följa Team Sportias profil.

2.1.2 Funktioner

2.1.2.1 Applikationen skall kunna stängas med en knapp

2.1.2.2 Applikationen skall kunna skriva ut kvitto

2.1.2.3 Samtliga fält skall kunna rensas ned en knapp

2.2 Datakrav

2.2.1 Indata

2.2.1.1 Vid registrering anger man kundens personnummer

2.2.1.2 Vid registrering anger man cykelns EAN nummer

2.2.1.3 Vid registrering anger man eventuellt kampanjpris

2.2.1.4 Vid registrering anger man eventuellt tillbehörspris

2.2.1.5 Vid registrering anger man cykelns ramnummer

2.2.1.6 Vid registrering anger man cykelns S-REG nummer

2.2.1.7 Personnummer kan hämtas in med streckkodsläsare från kundens identitetshandling

2.2.1.8 EAN nummer kan läsas in med streckkodsläsare från cykelns informationslapp

2.2.1.9 Ramnummer kan läsas in med streckkodsläsare från cykelns informationslapp

2.2.1.10 S-REG nummer kan läsas in med streckkodsläsare från cykelns informationslapp

2.2.2 Utdata

2.2.2.1 Förnamn hämtas vid korrekt inmatat personnummer

2.2.2.2 Efternamn hämtas vid korrekt inmatat personnummer

2.2.2.3 Adress hämtas vid korrekt inmatat personnummer

2.2.2.4 Telefonnummer hämtas vid korrekt inmatat personnummer

2.2.2.5 Förnamn visas efter hämtning

2.2.2.6 Efternamn visas efter hämtning

2.2.2.7 Adress visas efter hämtning

2.2.2.8 Telefonnummer visas efter hämtning

2.2.2.9 Leverantör hämtas vid korrekt inmatat EAN nummer

2.2.2.10 Artikelnamn hämtas vid korrekt inmatat EAN nummer

2.2.2.11 Märke hämtas vid korrekt inmatat EAN nummer

2.2.2.12 Pris hämtas vid korrekt inmatat EAN nummer

2.2.2.13 Leverantör visas efter hämtning

2.2.2.14 Artikelnamn visas efter hämtning

2.2.2.15 Märke visas efter hämtning

2.2.2.16 Pris visas efter hämtning

2.2.2.17 När funktionen skrivut anropas skrivs ett garantikvitto ut på angiven standardskrivare

2.3 Garantikvitto

2.3.1 Utseende

2.3.1.1 All inhämtad data ska presenteras på kvittot

2.3.1.2 Försäljningsdatum ska finns med på kvittot

2.3.1.3 Vem det är som säljer ska finnas med på kvittot

2.3.1.4 Garantiinformation ska finnas med på kvittot

2.3.1.5 Team Sportias logotyp ska finnas med på kvittot

2.3.2 Funktion

2.3.2.1 Garantiinformationen hämtas från en fil

2.3.2.2 Garantiinformationen är olika för olika leverantörer

3 Kvalitetskrav

3.1 Utvidgbarhet

3.1.1 Utrymme för att spara köpet i stöldförsäkringsdatabas skall finnas

3.2 Prestanda

3.2.1 Plattform

3.2.1.1 Applikationen skall fungera på Microsoft Windows XP

3.2.1.2 Applikationen skall fungera på Microsoft Windows 2000

4 Projektkrav

4.1 Utvecklingsmiljö

4.1.1 Databaser

4.1.1.1 Oracle 10g

4.1.1.2 IBM DB2

4.1.2 Applikationer

4.1.2.1 Microsoft Visual C++ 2005

4.1.2.2 Adobe Photoshop CS2

4.1.2.3 Microsoft Excel 2004 (Mac)

4.1.2.4 Microsoft Word 2004 (Mac)

4.1.2.5 Apple Keynote 3.0.1

4.1.2.6 Visual C++.NET

4.1.3 Programspråk

4.1.3.1 C++

4.2 Leveranskrav

Applikationen kommer endast att vara en prototyp
Vid leverans ska applikationen lämnas över på CD
Vid leverans ska nödvändiga systemfilspaket finnas med på CD

5 Terminologi

5.1 Från kravspecifikationen

5.1.1 EAN (European Article Number) är det nummer för en specifik artikel. Standard över hela världen

5.2 Från rapporten

5.2.1 ODBC (Open DataBase Connectivity)-koppling är en standardmetod för att ansluta till vilken databastyp som helst

5.2.2 Målsystem är Team Sportias egna datornätverk

5.2.3 Testmiljö avser den miljö vi använder vid utveckling av applikationen