HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i informationssystemutveckling, 20 p
C-nivå
Vårterminen 2005

Problem vid migration av applikationer från Visual Basic 6 till Visual Basic .net

vid användande av ett migrationsverktyg

Henrik Dalemo

HS-IKI-EA-05-403

Problem vid migration av applikationer från Visual Basic 6 till Visual Basic .net vid användande av ett migrationsverktyg

Examensrapport inlämnad av Henrik Dalemo till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information.

050604

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Handledare för examensarbetet: Henrik Gustavson

Examensarbete Problem vid migration av applikationer från Visual Basic 6 till Visual Basic .net vid användande av ett migrationsverktyg

Henrik Dalemo

Sammanfattning

Arbetet bygger på att de informationssystem som används idag men som blir allt äldre och därigenom inte klarar de krav som ställs på dem. Detta medför att informationssystemen måste migreras till modernare plattformar, då företagen dels inte klarar sig utan dess nuvarande funktionalitet och dels måste vidareutveckla systemen.

Detta arbete tar upp olika sätt som detta kan genomföras på och vilka problem som kan komma att uppstå när detta sker. För att kunna undersöka detta så genomfördes ett antal migrationer. Språken som migrationen kommer att ske mellan är Visual Basic 6 och Visual Basic .net. Migrationerna kommer att ske med hjälp av ett migrationsverktyg, samt ett ramverk som i arbetet skräddarsys för just arbetets typ av migrationer.

Det arbetet kommer fram till är att det sker problem vid användande av ett verktyg och även att det inte ger tillräckligt med respons på vad som gått snett under migrationerna.

Nyckelord: Migration, Ramverk, Databasapplikationer

Innehållsförteckning

1 Introduktion

IT-branschen är idag en bransch som snabbt växer och som snabbt förändras. Enligt Wu et. al. (1997) blir de äldre informationssystemen allt mer ett hinder i utvecklingen då de bland annat inte kan utföra den funktionaliteten som krävs för att hålla ett företag konkurrenskraftigt. Företagen står då inför problemet att de har ett föråldrat informationssystem men att de inte klarar sig utan den funktionalitet systemet erbjöd.

Problemen som kan uppstå är enligt Bisbal et. al. (1999) att de system som företagen har inte längre håller måttet gentemot de krav som ställs på dem. Det kan vara underhållskostnader som blir för höga eller att hårdvaran inte klara av de nya kraven. Den kan även vara svårt att bygga ut det gamla systemet och intrigera det med nya applikationer. Något som är vanligt idag är att företag slås ihop eller att mindre företag blir uppköpta. De system som de båda företagen har måste då kunna integrera med varandra. Något som vissa språk inte klarar av lika bra som andra.

För att hitta en lösning på dessa problem väljer företag ibland att migrera kod. Det vill säga flytta över systemets funktionalitet och data till en annan plattform. Problemet är att detta inte sker smärtfritt. Det är heller inte klart vilken plattform som ska användas eller vilket språk som de nya applikationerna ska programmeras i. Ett annat problem som företagen ställs inför är hur själva flytten eller migrationen av applikationer och data ska ske.

Det detta arbete fokuserar på är hur migrationen av applikationer kan ske med hjälp av ett migrationsverktyg och vilka, om några, problem som uppstår i och med detta. Experiment av migrationer kommer att genomförs för att få fram dessa problem. Migrationen kommer att ske från Visual Basic 6 till det nyare Visual Basic .net med hjälp av det inbyggda migrationsverktyg som finns i Visual Studio .net. Utifrån de migrationer som genomförts under arbetets gång så tas dels de problem som uppstår upp och dels så skräddarsyddes ett ramverk för just den typ av migration som arbetet bygger på. Detta ramverk baseras på redan utvecklade ramverk som finns på marknaden idag.

Rapporten visar att det ofta blir fel vid migration av det migrationsverktyg som Microsoft tagit fram och som används i arbetet. Det blir ofta designfel och i vissa fall faller funktionaliteten när migrationen genomförs. Ramverket som tagits fram fungerar bra och kan vara till stor hjälp vid denna typ av migrationer.

2 Bakgrund

Här följer en introduktion om vad en RADmiljö (Rapid Application Development) är för att sedan gå in mer i detalj och se på olika slags RADmiljöer och de språk som används i dessa. Det finns även ett avsnitt om vad de olika miljöerna använder för verktyg för att hämta data från databaser och hur dessa verktyg används. Sist kommer ett avsnitt som tar upp olika sätt att migrera kod mellan olika plattformar och språk samt exempel på olika ramverk för detta.

2.1 Utveckling med RAD miljöer

Exakt vad en RADmiljö är för något finns det många olika svar på. Price (2003) beskriver en RADmiljö som något som gör det möjligt att utveckla, köra och felsöka program i en integrerad utvecklingsmiljö. Även Dietel et.al (2002) beskriver RADmiljöer på liknande sätt genom att säga att det på ett bekvämt sätt går att skriva, köra, testa och felsöka program, och fortsätter med att säga att detta innebär att tiden som måste läggas på utvecklingsarbete blir mindre. Dietel et. al (2002) säger även att det är själva processen i arbetet som betecknar utveckling i RADmiljöer och att Visual Basic är världens största RADspråk.

Exempel på olika RADmiljöer är Access (Dobson, 2001), Deplhi (Cantù, 2003) samt Visual Basic (Dietel, 2002). RADmiljön hjälper framförallt användaren med att programmera gränssnittet. Det är dock inte möjligt för användaren att helt undvika att programmera själv, det vill säga skriva kod manuellt, särskilt vid mer avancerade applikationer. Hur mycket hjälp användaren får beror på vilken miljö som används. Om en jämförelse genomförs mellan #-Develop, som är RADmiljön för C#, och den nyaste versionen av Visual Studio (Visual Studio .net), som är en av de absolut populäraste och mest använda miljöerna just nu, så ger det sistnämnda mest stöd åt användaren (Jonsson, 2003).

Visual Studio .net är den miljö som Microsoft har utvecklat för att kunna göra .net applikationer. I denna miljö går det att använda en rad olika språk bland annat Visual Basic .net och C# som är två av de mest omtalade språken just nu (Abramson & Watsson (2003), Heasman, (2004)). Enligt Jonsson (2003) så sker utvecklingen av en applikation ofta i tre steg när RADmiljöer används. Första steget är att göra bakgrunden i formuläret, ta fram bakgrundsmotiv/färg, storleken på formuläret och så vidare. I steg två lägger utvecklaren till de knappar, listor och andra kontroller som kommer att behövas i formuläret. I det tredje och sista steget kommer programmeringen in, här bestäms vad de olika kontrollerna ska göra. Stegen utförs oftast iterativt så att utvecklaren bygger på mer och mer på applikationen för att lättare kunna hitta fel och liknande i programmet då de delar som fungerat innan kan uteslutas om ett fel uppstår.

Det är den första och andra fasen som RADmiljöerna skiljer sig från textbaserad programmering då utvecklaren här använder sig av ”klippa och klistra” för att utforma designen på applikationen. Genom ”klippa och klistra” metoden placeras olika knappar och listor in i formuläret genom användning av musen, vilket innebär att det inte behövs skrivas någon kod för just detta. Genom att jobba på detta visuella sätt är det enligt Jonsson (2003) lättare för utvecklaren att direkt se hur den färdiga applikationen kommer att se ut och behöver därigenom inte planera så mycket i förväg hur den ska se ut.

Det finns tre tillstånd som Visual Studio .net kan befinna sig i när det används. Detta för att utvecklaren ska kunna utforma och koda, för att sedan kunna rätta koden och till sist exekvera applikationen. Det första tillståndet utvecklaren kommer i kontakt med är designläget. När arbetet i detta tillstånd gjorts finns det i Visual Studio. net ett debuggläge som inte alla andra miljöer har där utvecklaren kan göra en felsökning i koden på en applikation innan den exekveras och hamnar i exekveringsläget (Jonsson, 2003).

2.2 RADmiljöer

I detta avsnitt så kommer det att tas upp en kort beskrivning av två av de vanligaste programmeringsspråken inom RADmiljöer samt en presentation av de miljöer som används vid programmering med dessa språk. Språken som tas upp är Visual Basic 6 och dess uppföljare Visual Basic .net. Det som kommer att tas upp är grundläggande bakgrundsinformation om språken som används och hur de förhåller sig till varandra. I nästa avsnitt ”databasaccess i RADmiljöer” så kommer det beskrivas hur kopplingar till databaser sker inom de olika miljöerna.

2.2.1 Visual Basic

Visual Basic är framtagit av Microsoft och är en vidareutveckling från språket BASIC som kom under 60-talet (Perry, 1999). BASIC var ett rent textbaserat språk som skulle göra det lättare för nybörjare att börja programmera. BASIC utkom i en mängd olika versioner, bland annat QBASIC, BASICA och MBASIC. Alla dessa versioner var textbaserade vilket innebär att de endast går att köra i texteditorer och alltså endast visa text, medan nyare språk som Visual C++ har ett gränssnitt där användaren kommunicerar med programmet via bilder och figurer likt det windowsgränssnitt som idag är vanligt. För att kunna köra BASICprogram i windowsmiljön introducerade Microsoft Visual Basic, vilket grundar sig på BASICspråket men körs visuellt. Detta innebär att programmen både utvecklas och visas visuellt genom Visual Studio som är den RADmiljö som Visual Basic använder sig av. För mer avancerade Visual Basic applikationer måste användaren gå in och manuellt skriva kod då all funktionalitet inom språket inte går att programmera visuellt (Perry, 1999).

Skillnaden mellan Visual Basic och andra visuella språk som Visual C++ är, enligt Perry (1999), bland annat att det finns en tolkningsfunktion. Med denna funktion kan utvecklaren se om programmet innehåller några programmeringsfel samt vad som är fel i det redan innan det körs, genom att funktionen söker upp buggar i programmet. Detta innebär att det är lättare för programmeraren att direkt se hur programmet kommer att fungera. Att språket är tolkande är framförallt till fördel för nybörjare som med en gång ser vad de gjort fel och lättare se vart felen uppstår. Visual Basic går liksom andra språk att kompileras. På grund av den ökade prestandan vid kompilering är detta viktig när programmen ska användas kommersiellt (Perry, 1999). Att de går att kompilera gör Visual Basic till mer konkurrenskraftigt gentemot mer avancerade språk. Huruvida applikationerna ska tolkas innan de kompileras är upp till utvecklaren men enligt Perry (1999) så sker det med fördel efter att tolkningsfuntionen genomförts, även för mer erfarna programmerare.

2.2.2 Visual Basic .net

.net introducerades 2002 av Microsoft. Den övergripande tanken med .net är att program ska bli mer Internetbaserade och vara mer åtkomligt från olika apparater som mobiltelefoner, bärbara datorer med mera. Med .net är det möjligt att kombinera

applikationer skrivna på olika språk som Visual Basic.net eller C#. Detta ger programmen större möjlighet att kunna kommunicera med varandra oberoende vilken plattform som de körs på. En stor del i detta har Web services som i och med .net även kan köras med Visual Basic språket. En webb service gör att applikationer via standardiserade språk, som XML, kan kommunicera direkt med varandra över Internet. .net kan ses som ett operativsystem som ligger ovanpå Windows som i stort består av en runtimemiljö och ett objektorienterat klassbibliotek. Klassbiblioteket erbjuder ett grafiskt gränssnitt för exempelvis windowsapplikationer, webbapplikationer och databaser (Mössenböck et. al. 2003).

Det som gör det möjligt för olika språk att fungera ihop inom .net är runtimemiljön, Common Language Runtime (CRL), som även ger större säkerhet och bättre skräphantering. CRL liknar Javas virtual machine men skillnaden från Java ligger i just det att det finns stöd för flera olika språk (Heasman, 2004). Detta sker genom att koden som skrivs översätts till det så kallade Common Intermediate Language (CIL) just innan programmet körs. För att möjliggöra detta var det tvunget att alla språk som skulle stödja .net skulle ha samma sorts datatyper. Hur en datatyp i .net ska se ut bestäms av Common Type System (CTS) som har till uppgift att visa standarden på datatyperna. I och med detta kan en klass som programmerats i exempelvis C# köras i alla andra program så länge de stödjer .net (Mössenböck et. al. 2003).

Visual Basic .net är en utveckling från Visual Basic 6 designat för att kunna programmera applikationer för .net miljön. Det är designat för att utvecklaren snabbt och enkelt ska kunna skapa applikationer (Fedorov, 2002). Förändringarna som gjorts sedan version 6 av Visual Basic är stora och dessutom på betydande områden av språken, det är därför ansett att bytet från Visual Basic 6 till Visual Basic .net är mer än ett versionsbyte (Mackenzie, Sharkey, 2002). En av anledningarna till de stora förändringarna är just bytet till .netmiljön, men även att Microsoft har tagit bort mycket av de funktioner som funnits med språket sedan det introducerades och som byggts på under tidens gång men som nu blivit förlegade, detta har resulterat i en renare version av språket.

Utvecklingsverktyget, Visual studio, som användes till Visual Basic 6 är även det ändrat och heter nu Visual Studio .net. I denna miljö går det att använda sig av en mängd olika språk, exempelvis, C#, J# Visual Basic .net eller C++. Alla språken körs nu genom en gemensam exekveringsmiljö kallad CLR (Common Language Runtime). Syftet med detta är att det ska gå att köra så många språk som möjligt genom just Visual Studio .net, vilket har resulterat i att Visual Basic .net har blivit mer likt andra mer objektorienterade språk och därmed även att kodsyntaxen har ändrats något jämfört med tidigare versioner av Visual Basic (Jonsson, 2002).

Vidare har det nya Visual Basic .net fått förbättrade eller helt nya funktioner inom dess gränssnitt och arv. Det har även fått en mer fri trådningsfunktion och en mer strukturerad funktion för minneshantering. På grund av att språket är integrerat med .net ramverket och som tidigare nämnts exekveringsmiljön CRL har språket även blivit mer säkert, fått bättre skräphantering och även bättre support (Fedorov, 2002).

2.3 Databasaccess i RADmiljöer

I detta avsnitt kommer dataaccess i dels Visual Basic 6 och dels i .net miljön att beskrivas. Det som kommer att tas upp är tekniken bakom databasaccessen och inte så

mycket just exakt hur det med kod genomförs i de olika miljöerna. Det kommer att bli en övergripande jämförelse mellan den gamla miljön och det nyare .net för att öka förståelsen av de olika teknikerna och hur de jobbar.

2.3.1 Databasaccess i Visual Basic

Enligt Jonsson (2002) finns det i Visual Basic två grundläggande sätt att komma åt och hantera data i en databas. Kopplingen till en databas i Visual Basic sker via en objektmodell. Version 6 av Visual Basic har två olika objektmodeller. Den första som lanserades kallas DAO (Data Access Objects) och den nyare kallas ADO (ActiveX Data Objects). ADOtekniken använder sig av OLEDBprovidrar som fungerar som ett mellanskikt mellan ADO och data i databaserna. ADO är enligt Jonsson (2002) mer komplicerad rent tekniskt att upprätta än DAO.

När användaren ska ändra, söka eller lägga till tabeller i databaser finns det ett inbyggt verktyg i de flesta versioner av Visual Basic 6 kallat Visual Data Manager. Detta verktyg går att använda utan den mer avancerade ADOmodellen. Perry (1999) beskriver dock verktyget som ”extremt begränsat”. Som exempel tar han upp att det inte går att skapa rapporter eller formulär för användaren. Det är mer ett verktyg för att analysera databasfiler än ett databassystem, inte alls i klass med exempelvis Access eller FoxPro. På grund av just dess begränsning rekommenderar Perry (1999) inte att skapa en hel databas för en organisation med denna funktion. När det gäller att arbeta mer avancerat med databasen rekommenderar Perry (1999) istället de användare som inte har tillräcklig kunskapen om ADO att använda den så kallade datakontrollen. Detta verktyg är mer avancerad än Visual Data Manager. Denna finns dock bara i vissa versioner av Visual Basic 6.

När det gäller riktig avancerat arbete med databasen måste användaren enligt Perry (1999) ta hjälp av ADOmodellen. Fördelen med att använda den mer avancerade ADOmodellen är bland annat att den har bättre förutsättningar när det gäller att koppla till databaser över Internet eller andra typer av nätverk, den har även en snabbare åtkomst till databasen. Vid användning av ADOmodellen går det även att få mer alternativ när det gäller dataåtkomsten från databasen, samt att hantera mer komplexa saker som E-post-text, bilder och webbläsare, medan DAO endast hanterar databaser. Det går att använda båda modellerna i samma applikation, det som avgör är oftast hur komplicerade de ska vara.

2.3.2 Databasaccess i .net

För dataaccess i .net används inte längre den version av ADOmodellen som var den huvudsakliga användningen för dataaccess i Visual Basic 6, istället har ADO.NET tagits fram som är en nyare variant av den tidigare ADOmodellen. Både ADO och ADO.NET använder sig bland annat av OLEDBklasser för att kommunicera med databaserna. De båda ADOmodellerna är tekniskt sett lika varandra men enligt Mössenböck et. Al. (2001) har ADO.NET ett par speciella klasser för åtkomst av en SQLserver. Det går dock att komma åt en SQLserver även genom den gamla ADOtekniken genom de olika OLEDBklasserna som finns. Det går även att ta fram två olika typer av formulär i ADO.NET, windowsformulär och Internetformulär.

Ytterligare en skillnad mellan de två ADOteknikerna är enligt Price (2003) att .net kan hämta data som är helt frånkopplad från databasen, eller vilken källa den nu har, genom DataSets. Detta gör att applikationen kan arbeta med data under en längre tid utan att den behöver vara uppkopplad mot databasen under hela tiden. Den blir då

även oberoende av vilken källa data kommer ifrån. Den äldre versionen av ADO har en liknande funktion som kallas recordset, men dessa två tekniker skiljer sig något från varandra. Datamängderna kan hämta information från många olika tabeller, hitta relationer mellan dessa samt att skapa många olika vyer på en och samma tabell. Datamängderna blir mer som små databaser som finns tillgängliga i minnet.

I ADO.NET har även vyerna och databindningen utvecklats ytterligare. Det går nu att skapa en sorteringsordning på vyerna som kan användas till filtrering av rader. Just filtreringen går att basera på fyra olika huvudsätt. Filtrering på vilka som har ändrats och inte samt vilka som tagits bort och lagts till. När det gäller databindning så har ADO.NET utvecklats så att det även går att binda tabeller, vyer, datamängder matriser. I ADO gick det endast att binda DataSetObjekt. Med bindning menas den koppling som gör det möjligt att via gränssnittet ändra och visa data från databaser.

2.4 Migration

Bisbal et. al. (1999) tar i en artikel upp problem med att underhålla, förändra och bygga ut så kallade Legacy informationssystem. Artikeln ger exempel på framförallt tre olika lösningar på problemet, dessa presenteras här nedan. Anledningen till att vilja ändra i sitt informationssystem kan enligt författarna vara att hårdvaran som systemet körs på inte duger till och därför behöver bytas ut, kostnader för att underhålla den mjukvara som redan nu finns, det gränssnitt som systemet körs på är svårt att integrera med andra system och till slut tar de även upp problemet med att bygga ut just Legacy informationssystem. Även fast just dessa problem inte alltid stämmer överens med alla informationssystem eller applikationer som på något sätt behöver förnyas ger det ändå en bra bild över vad som skulle kunna vara anledningen till att förnya mjukvara.

2.4.1 Metoder för migration

Tre huvudsakliga metoder finns enligt Bisbal et. al. (1999) för att lösa problem med att förnya mjukvara. Med omimplementation går det att göra mindre ändringar från den ursprungliga applikationen men det är även den lösningen som medför mest risker. Utvecklarna börjar från grunden och bygger en helt ny mjukvara. Den som vanligtvis för tillfället medför minst arbete och som inte medför allt för stora risker är så kallad wrapping som bygger på att ta ut vissa komponenter som fungerar bra och enbart ge den en ny och bättre mjukvara. Det mellersta alternativet är migration, som här menas den specifika metod som ändvänds och inte den mer övergripande betydelse som avser att underhålla, förändra och bygga ut system. I resterande delar av arbetet är det den mer specifika betydelse som avses. Vid migration flyttas funktionen och data från det gamla systemet med över till en ny plattform som bättre passar i den nuvarande situationen, eller i en framtida.

Vilken av dessa tre metoder som ska väljas beror enligt Bisbal et. al. (1999) på situationen som råder, det går inte att säga att en metod alltid fungerar bäst. Det går heller inte att säga att det bara ska användas en metod, olika delar av mjukvaran kan vara mer lämplig till olika metoder. Metoden omimplementation sker med en bigbangimplementation där hela systemet måste stängas ner för att kunna få in de nya implementationen. Denna metod anses vara den mest komplexa. Basen i denna metod är att från grunden bygga ett nytt system med ny plattform, arkitektur, verktyg och databaser. Artikeln tar upp två sätt att införa omimplementation på men säger samtidigt att inte någon av dem skulle fungera särskilt bra i praktiken. Den första bygger på att dela in de problem verksamheten har med sitt system i olika faser, för

att sedan gå in mer specifikt i varje problem. Detta sätt anses ligga på en för hög nivå för att kunna ge tillräckligt med hjälp i en praktisk situation. Det andra sättet som tas upp bygger på att decentralisera systemet genom att utveckla separata processer som det byggs applikationer till och som sedan kopplas samman. Detta är tänkt att ge en mjukare övergång till det nya systemet, men det faller på att det inte står riktigt klart hur detta skulle kunna genomföras. Generellt är omimplementation väldigt riskfyllt, det tar dessutom tid vilket kan resultera i att verksamheten har förändrats för mycket under tiden den nya implementationen utvecklas.

I andra ändan av komplexitetsnivån ligger wrapping. Idén bakom wrapping är enligt Bisbal et. al. (1999) att ge de gamla delarna i systemet ett nytt ansikte genom att implementera ett så kallad GUI (Graphical User Interface). Användarna använder sig då av det nya gränssnittet vilket även, till en viss gräns, kan ge nya funktioner. Men grunden är den samma vilket resulterar i att många av de gamla problemen kvarstår. Vissa saker, som underhåll, kan till och med bli svårare att hålla koll på då det läggs till ytterligare en komponent till systemet. Att använda wrapping för att ge sitt system ett nytt gränssnitt kallas screen scrapping och är enligt Bisbal et. al. (1999) det mest använda. Wrapping anses vara en kortsiktig metod som enbart erbjuder tillfälliga lösningar, de stora problemen kvarstår eller kommer snart åter upp till ytan. Fördelarna med den är att den är billig och går fort att genomföra.

Den tredje och sista metoden som ligger mellan de två tidigare som tagits upp är migration. Migration passar sig som bäst när omimplementation är för riskfyllt och wrapping på ett eller annat sätt inte passar. Även om migration är väl utbrett och ofta används finns det få approacher på hur verksamheterna ska gå till väga med migrationen. Bisbal et. al. (1999) framhäver även att det inte finns en approach som passar alla situationer och system utan att detta måste regleras. Det som bör finnas är några få övergripande stöttepelare att hålla sig efter. Vid förarbetet till en migration är det viktigt att beslutsfattarna vet vilka problemen är och vad som ska göras för att få bort dessa.

Tanken bakom migration är att flytta systemet till en annan plattform (Bisbal et. al. 1999) utan att det ska påverka funktionaliteten i själva systemet. Om det läggs in nya funktioner i systemet under migrationen kan det bli svårigheter att kunna säkra att systemet kan utföra samma saker som innan. Om det behövs ytterligare funktioner i systemet ska dessa implementeras innan eller efter migrationen har genomförts. Att lägga till nya funktioner sker ändå ofta då det är lättare att få migrationen finansierad på det sättet. Migration lägger stor vikt vid att få så lite störningar som möjligt vid själva överflytten till den nya plattformen. Det är viktigt att utvecklarna väl känner till det system som de nu har för att kunna återskapa detta i den nya miljön. De funktioner och den output som systemet genererar måste stämma överens med vad det gamla systemet genererade. För att försäkra sig om detta undersöks systemet innan det börjar användas.

För att se till att data kommer att kunna användas är det viktigt för utvecklarna att jämföra den nya och den gamla databasen för att se om data behöver transformeras på något sätt innan den förs in i det nya systemets databas. Utvecklarna kan välja att ta olika sektioner av data och föra över istället för all data på en gång. Det är även viktigt att data som skickas över är ren annars måste den tvättas, detta kan göras under migrationen men även före eller efter själva överförandet. (Bisbal et. al. 1999)

När det väl kommer till själva överförandet, eller cut-over som det kallas, finns det tre huvudsakliga linjer att gå efter. Den första, *Cut-and-run*, förespråkar att helt enkelt sluta använda det gamla systemet och direkt gå över till det nya systemet. Detta är en väldigt riskfylld lösning då det inte är säkert att allt fungerar och att vissa saker kan ta tid innan de upptäcks. Om allt flyter på som det ska är det en billig och snabb lösning men chansen att det inte sker komplikationer är väldigt liten varpå denna linje inte är att rekommendera, i alla fall inte vid större system. Vid tillämpning av den andra linjen, *Phased Interoperability*, så sker övergången till det nya systemet stegvis, mindre delar eller applikationer byts ut i omgångar för att då inte riskera att allt ska braka samman. Problemet är att det blir en väldigt komplex situation då det är svårt att få de olika delarna att fungera separat såsom de gör tillsammans och även data som används under övergångsperioden måste kunna fungera tillsammans. Den tredje och sista linjen, *Pararell Operations*, går ut på att under en tid använda sig av båda systemen för att försäkra sig om att det nya fungerar och ha det gamla i backup om det nya skulle krascha eller att det skulle upptäckas fel i det. Under denna period är det viktigt att testa att systemet fungerar som det ska, annars är det ingen mening med att köra systemen pararell. Det är svårt att ta ut den bästa av dessa tre och ständigt gå på den linjen, då det blir olika situationer i varje migrationsprojekt. Det som Bisbal et. Al. (1999) ger som förslag är en kombination av de olika för att det ska kunna ske smärtfritt, då själva cut-over fasen är viktig i migrations projekt.

2.4.2 Ramverk vid migration

Det finns en rad olika ramverk för hur dessa metoder som presenterats ska utföras. Ramverken passar mer eller mindre till de olika metoderna. De vanligaste ramverken är Chicken Little, Cold Turkey och Butterfly. För omimplementation nämns ofta Cold Turkey som ett alternativ (Bisbal et. al. 1999). Cold Turkey lämpar sig bra till den bigbangapproach som används vid omimplementation. Brodie & Stonebraker (1993) beskriver ramverket enligt följande ”Cold Turkey involves rewriting a legacy IS from scratch to produce the target IS using modern software techniques and hardware of the target environment". Cold Turkey har dock fått en hel del klagomål på sig för att det är för stora risker med detta ramverk. Det två andra passar bättre till migrationsmetoden (Bisbal et. al., 1999). Chicken Little går ut på att, i 11 faser, migrera vissa delar av systemet för sig. Medan migrationen sker kommunicerar de två systemen (det äldre och det nya) med varandra genom gateways (Brodie & Stonebraker 1995). I det tredje ramverket, Butterfly, finns inte dessa gateways. Överföringen sker här istället på så sätt att data sparas ner och förs över till det nya systemet under migrations fasen, och alla ändringar som ska ske på data under tiden som migrationen sker loggas för att senare föras in i systemet (Wu et. al. 1997). Även detta ramverk delas in i olika faser.

För den typ av migrationsprojekt som detta arbete inriktar sig på så är de två metoderna Chicken Little och Butterfly de som lämpar sig bäst. Här nedan kommer därför dessa två att presenteras grundligare. Ramverket Cold Turkey bygger enligt Bisbal et. al. (1999), som tidigare nämnts, mest på omimplementation av systemet och inte migration som kommer att användas i arbetet. Brodie & Stonebraker (1993) listar dessutom en hel del negativa saker med just Cold Turkey. Bland annat tar de upp att det finns en stor risk att migrationen kommer att misslyckas på grund av att det måste till en förändring av systemet i sig för att en styrelse ska gå med på en migration med hjälp av Cold Turkey då det medför så stora risker. Cold Turkey har heller ingen teknik för Cut-over fasen då bytet mellan det äldre och det nya systemet sker vilket gör att systemet inte kan utnyttjas under tiden migrationen sker, något som kan vara

svårt att få till då det kan ta en lång tid att göra överförandet. Cold Turkey har heller ingen metod för att kunna dela upp systemet som ska migreras i mindre delar utan utgår från att hela ska migreras på en gång vilket kan leda till en väldigt komplex migration. Detta gör att Cold Turkey inte ansågs vara tillräckt passande för detta arbete vilket medförde att det valdes bort.

2.4.2.1 Chicken Little

Chicken Little är ett ramverk som är tänkt att användas vid migration av större system. Ramverk delas in i 11 steg där beslut om hur steget ska utföras måste tas i varje steg och där besluten även kan påverka de andra följande stegen. Ramverket bygger på att dela in systemen i mindre delar för migration (Brodie & Stonebraker 1995). Det nya systemet är alltså från början endast en eller ett par applikationer för att sedan byggas på en efter en. I detta ramverk kan verktyg användas om det anses som en fördel.

För att utföra själva överflytten används gateways som fungerar som en bro mellan det äldre och det nya systemet. Gateways fungerar så att alla ändringar som genomförs från det äldre systemet transformeras och plockas upp för att kunna distribueras till andra ställen. En gateway kan även ändra data i det äldre och det nya systemet så att båda kan utnyttjas vid sidan om varandra under en period när skiftet mellan de två sker (Brodie & Stonebraker 1995). Här följer en mer ingående beskrivning av ramverket.

Steg 1: Analysera ursprungskoden: Enligt Brodie & Stonebraker (1995) ska utvecklaren kontrollera sin kod och se vad som krävs av den. Många av de krav som tidigare fastställts kan vara föråldrade eller i nuläget helt irrelevanta. Systemet ska återutvecklas tillbaka till kravfasen i utvecklingsarbetet för att kunna få fram specifika krav. Kraven som kommer att tas fram till det äldre systemet kommer även att ställas på den nya.

Steg 2: Bryta ner ursprungssystemets struktur: Här gäller det enligt Brodie & Stonebraker (1995) för utvecklaren att se till att systemet kan brytas ner i mindre delar, vilket är Chicken Littles signum. Delarna kommer sedan var för sig att migreras vidare. Det är viktigt att det inte finns några kopplingar mellan delarna såsom procedurer som ropar på varandra och liknande. Dessa måste tas bort innan migrationen kan börja. Det är även viktigt med ett väldefinierat gränssnitt mellan modulerna och databasen.

Steg 3: Designa det nya gränssnittet: Gränssnittet för det nya systemet ska se så likt ut det för den äldre såvida inga nya funktionaliteter har lagts på, vilket inte rekommenderas under migrationen. Här används ofta så kallade GUIs. Även migrationen av gränssnittet ska planeras i detta steg, exempelvis huruvida en gateway ska användas för att kunna samköra de båda systemen eller inte (Brodie & Stonebraker, 1995).

Steg 4: Designa den nya applikationen: Det finns två sätt att designa de nya applikationerna på, antingen ska de designas så att de bäst passar den nya miljön alternativt designas så att de blir så likvärdiga de ursprungliga applikationerna som möjligt. Detta är en balansgång av olika risker som kan uppstå. Ändras det i applikationerna är det inte säkert att funktionaliteten i det nya systemet är den samma

som i det äldre men om inga ändringar genomförs utnyttjas troligen inte den nya miljön till fullo. Den analys som genomfördes i steg ett används som underlag till detta (Brodie & Stonebraker, 1995).

Steg 5: Designa den nya databasen: Den nya databasen ska kunna möta de krav som ställts på systemet. En rekommendation är ändå att använda sig av en relations-SQLdatabas så långt det går. Detta steg i processen kan vara väldigt enkelt om databasen för det äldre systemet till stor del kan användas i den nya miljön, men steget kan även vara mycket komplext då databasen behöver mycket ändringar för att kunna möta kraven från system. Det gäller att det har gjorts en bra analys av den äldre databasen så allt relevant kommer med i den nya. Utvecklaren måste även ha i åtanke huruvida grundtanken med databasen ska ändras eller inte, ska den exempelvis vara centraliserad eller decentraliserat? Även här är det risker och fördelar kontra nackdelar som bestämmer (Brodie & Stonebraker, 1995).

Steg 6: Installera den nya miljön: Här är det vikigt att se till kraven på hela systemet som ska installeras. Det är att föredra att se den nya miljön som helt ny och inte som en miljö att migrera till så att även de problem som inte hör till migreringen kontrolleras. Efter installationen är det viktigt att systemet testas noggrant innan det börjar användas för sig självt. Det är viktigt att se till att data blir rätt och prestandan håller tillräckligt hög nivå (Brodie & Stonebraker, 1995).

Steg 7: Utveckla och installera nödvändiga gateways: Det kan behövas en eller flera gateways till migrationen. Detta beror bland annat på hur systemet ser ut och på vilken plattform det körs på och ska migreras till. Antingen utvecklas gatewayen på plats eller så köps en in. För att kunna bestämma vilket som är bäst ses det till dess arkitektur, placering och krav som ställs på den då (Brodie & Stonebraker, 1995).

Steg 8: Migrera den ursprungliga databasen: Efter installationen av den nya databasen måste all data flyttas över från den äldre databasen. Detta sker ofta med hjälp av de gateways som nu finns på plats. För att kunna göra detta måste data tvättas, transformeras och laddas upp och ner. Detta kan ske i små grupper eller allt på en gång beroende på hur mycket data som ska migreras (Brodie & Stonebraker, 1995).

Steg 9: Migrera den ursprungliga applikationen: I detta steg skrivs de moduler som fanns i ursprungssystemet om för att passa den nya databasen, såvida inte det har bestämts att nya applikationer ska byggas. Det går att migrera en eller flera applikationer samtidigt men ju fler applikationer desto mer komplex blir migrationen. Det som bör beaktas är att det kan ta mer tid att migreras en och en. De viktigaste applikationerna migreras med fördel först, men det kan även vara en bra idé för de med lite erfarenhet av migration att börja med dem som verkar lättast (Brodie & Stonebraker, 1995).

Steg 10: Migrera det nya gränssnittet: Om ett 4GL som stödjer applikations och gränssnittsdesign används kan med fördel applikationerna och gränssnitten migreras tillsammans. Liksom vid applikationerna kan ett eller flera gränssnitt migreras samtidigt. Med hjälp av gateways kan systemet köras både från de nya och äldre gränssnitten under en tid då det nya systemets prestanda testas (Brodie & Stonebraker, 1995).

Steg 11: Övergång till det nya systemet: När det gäller övergången från ett system till ett annat finns det olika tillvägagångssätt att utföra detta på. Chicken Little har ett rätt fritt tillvägagångssätt för detta. Men ser helst att det sker applikation för applikation. Det kan även vara så att övergången sker vid olika tidpunkter för olika användare, särskilt vid system med stort antal användare. Detta steg kan även ske för de färdiga applikationerna innan nästa applikation börjat migreras (Brodie & Stonebraker, 1995).

2.4.2.2 The Butterfly Methodology

Det som enligt Wu et. al (1997) Butterfly i huvudsak bygger på är att användarna inte ska behöva köra både det äldre och det nya systemet samtidigt under den tid då själva överföringen sker av systemet. Butterfly skiljer på framtagandet av det nya systemet och datamigrationen vilket medför att de gateways som är viktigt i exempelvis Chicken Little ramverket utesluts helt.

När migrationen av data från det äldre till det nya systemet sker fryses data så att det endast går att läsa den, vilket resulterar i att inga ändringar av data sker under migrationsprocessen. Dessa frysta data sparas för att sedan kunna transformeras. Alla ändringar som genomförs mot systemet sparas istället ner i temporära filer för att sedan användas. Detta sker med hjälp av en så kallad Data Access Alocator (DAA), som fångar upp de kommandon som kommer och för den till den temporära filen istället. Med hjälp av en transformator, kallad Chrysaliser, överförs data från det äldre till det nya systemet. Efter alla data har transformeras börjar DAAn att spara alla inkommande kommandon i en ny temporär fil och Chrysalisern börjar transformera data från den första temporära filen. Detta sker iterativt tills det att storleken på den temporära filen är så liten att systemen kan vara nere under tiden transformationen från filen kan ske utan att systemen används. Ramverket delas in i sex olika faser för att bättre få en överblick på i vilken ordning detta ska ske. Faserna i sin tur består av olika aktiviteter (Wu et. al. 1997).

Fas 0: Förbered migrationen: I denna fas ska alla problem som kan uppstå vid migrationen tas upp. De viktigaste är dock användarkraven på dels migrationen och det nya systemet (Wu et. al. 1997).

0.1 Klargör de preliminära kraven.
0.1.1 Fastställ användarkrav.
0.1.2 Fastställ riktlinjer för mätning av hur bra migrationen gått.
0.2 Fastställ hur det nya systemet skall byggas upp.
0.3 Förbered hårdvaran systemet ska byggas på.

Fas 1: Förstå semantiken i det äldre systemet och utveckla det nya dataschemat*:* Här finns en hel del verktyg framtagna för att hjälpa utvecklaren men dessa rekommenderas att i så stor utsträckning som möjligt inte användas. Ett undantag är dock DAAn som används i aktivitet 1.6 (Wu et. al. 1997).

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.
1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ vilken data som ska migreras.
1.4 Identifiera och skaffa förståelse för kopplingar till andra system (valfri).
1.5 Slutför migrationskraven.
1.6 Utveckla DAAn.
1.7 Utveckla det nya dataschemat och fastställ kartläggningsreglerna.

Fas 2: Bygg upp ett exempeldatalager som, baseras på de exempeldata som tagits fram i det nya systemet: De främsta aktiviteterna i denna fas är att bestämma vilken data som ska andvändas som exempeldata och att utveckla Chrysalisern. Det är den data som ska hjälpa till att ta fram och testa det nya systemet (Wu et. al. 1997).

2.1 Fastställ exempeldata.
2.2 Utveckla Chrysalisern.
2.3 Transformera äldre exempeldata till ny exempeldata för det nya systemet och konstruera exempeldatalagret.

Fas 3: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen: Exempeldatalagret som konstruerades i fas 2 kommer att användas här för att kunna designa, utveckla och testa det som utvecklas i denna fas (Wu et. al. 1997).

3.1 Migrera det äldre gränssnittet.
3.1.1 Migrera/Utveckla en del av det nya gränssnittet.
3.1.2 Testa gränssnittet mot exempeldata.
3.1.3 Bekräfta gentemot användarkraven (valfri).
3.2 Migrera de äldre applikationerna.
3.2.1 Migrera/utveckla en applikation.
3.2.2 Testa applikationen mot exempeldata.
3.2.3 Bekräfta gentemot användarkraven (valfri).
3.3 Migrera återanvändbara äldre komponenter.
3.4 Integrera de nya komponenterna/systemen.
3.5 Testa de nya komponenterna/systemen för fel.
3.6 Bekräfta att de nya komponenterna/systemen stämmer överens med kraven.
3.7 Träna användarna på de nya komponenterna/systemen (valfri).

Fas 4: Migrera gradvis äldre data till det nya systemet och träna användarna i det nya systemet: Denna fas är framförallt till för att migrera de äldre data och är huvudfasen i Butterfly ramverket. Det är här de temporära filerna kommer att användas. Även DAAn och Chrysalisern kommer att användas (Wu et. Al. 1997).

4.1 Inkorporera DAAn i det äldre systemet.
4.2 Skapa de temporära filerna, och gör att det endast går att läsa från dessa.
4.3 Migrera alla data från den äldre till Exempeldatalagret 0 (EDL_0) med hjälp av Chrysalisern. Medan detta sker ska alla ändring av äldre data omdirigeras av DAAn till en ny temporär fil EDL_1, upprepa detta tills alla data har migrerats ur EDL_0.
4.4 Skapa EDL_2 och sätt EDL_1 till endast läsning.
4.5 Migrera alla data i EDL_1 till det nya systemet med hjälp av Chrysalisern. Omdirigera alla ändring av data med DAA_3 till EDL_2 tills EDL_1 helt har migrerats.

4.6 Repetera steg 4.4 och 4.5 tills alla data har migrerats tills EDL_n är tillräckligt liten för att systemen ska kunna frysas.
4.7 Frys hela det äldre systemet och migrera EDL_{n+1} till det nya systemet med hjälp av Chrysalisern.

Fas 5: Övergång till det nya systemet: Sista fasen, systemet ska nu vara klart att köras (Wu et. al. 1997).

5.1 Övergå till det kompletta nya systemet.

3 Problem

Problemet bygger på att det idag finns företag som använder sig av informationssystem kopplade till databaser men att dessa inte längre kan mäta sig med de krav som ställs på dem och därför behöver flyttas till en modernare plattform med större möjligheter. Problemet är hur migreringen av applikationer och data ska ske och att det ofta blir problem vid migreringen.

3.1 Problemområde

Trots att organisationerna är nöjda med de applikationer som används kan underhållskostnader öka med ett allt äldre system, hårdvaran klarar inte längre de krav som ställs på den eller att systemet måste få nya funktionaliteter som inte går att få på den äldre plattformen (Bisbal et. al. 1999). Det är då nödvändigt för organisationerna att byta upp sig till en ny plattform med större möjligheter. Framförallt är det inom interaktion som stora framsteg har tagits de senaste åren genom .net och framförallt web services.

Problemen med att byta plattform är att den äldre mjukvaran inte fungerar på den nya plattformen. Detta leder organisationerna till att migrera data, gränssnitt och applikationer från den äldre plattformen över till den nya. För detta finns en mängd olika tillvägagångssätt. Men frågan är hur detta ska ske på ett sätt så att både funktionaliteten och de data som finns i det nuvarande informationssystemet på ett säkert sätt kan föras över till den nya plattformen?

3.2 Problemprecisering

Det detta arbete kommer att inrikta sig på är migrering mellan Visual Basic 6 miljön till det nyare Visual Basic.net och vila problem som uppstår vid denna typ av migration. Anledningen till att just Visual Basic 6 valdes som exempel är att det är en av de mest använda miljöerna på senare år. Många av dess applikationer börjar bli begränsade när ny teknik börjar komma och Visual Basic .net miljön börjar växa sig allt starkare. Det kommer även att bli allt svårare att få support gällande Visual Basic 6. Det finns en mängd olika fördelar med Visual Basic .net jämfört med den äldre miljön som tas upp här, inte minst inom den ökade integrationen mellan olika applikationer med hjälp av web services. Det kan bli svårt för organisationer att inte halka efter i utvecklingen när nya metoder börjar komma som andra organisationer börjar använda sig av och tar för givet att andra också använder sig av dessa metoder.

Abramson & Watson (2003) tar upp problem med att företag måste ändra eller migrera kod från en plattform till en annan. De försöker att lösa problemen med hjälp av att automatiskt migrera koden från just Visual Basic till Visual Basic .net och sedan debugga den med en, vad de kallar, ”relative debugger” Detta sker dock inte helt smärtfritt.

Även Heasman (2004) tar upp problemet med att migrera kod mellan plattformar. “Invariably when migrating a large application from one platform to another there will be problems with legacy code.”

Problemet för detta arbete blir att ta fram vilka problem som uppstår vid migration från Visual Basic 6 till Visual Basic .net med hjälp av det migrationsverktyg som finns i Visual Studio .net, och att för detta ändamål skräddarsy ett ramverk till denna typ av migration.

För att kunna svara på dessa frågor kommer ett skräddarsytt ramverk för just migration från Visual Basic 6 till Visual Basic .net att tas fram och däri även gå igenom vad en utvecklare bör tänka på och vilka problem denne ställs inför när migrationen ska ske. Ramverket kommer att grunda sig i de metoder som redan idag finns, och sedan med hjälp av ett experiment kommer det att skräddarsys och ta fram riktlinjer till utvecklare.

3.3 Avgränsning

Arbetet utgår från att en migration ska ske. Ramverket som tas fram i arbetet täcker alltså inte hur frågan huruvida ett system ska migreras eller inte. Arbetet inriktar sig på att migrera med hjälp av migrationsverktyget som finns inbyggd i Visual Studio .net, vilken är den miljö som migreringen sker till. Som tidigare nämnts sker migrationer endast från Visual Basic 6 till Visual Basic .net.

Applikationerna är till största delen byggda på en databas som ligger på en lokal SQLserver. Båda miljöerna klarar av att hämta data från denna typ av databas varpå det därför inte tas upp hur data från en typ av databas till en annan skulle kunna ske.

Inte heller hur övergången från ett äldre till ett nyare system ska ske tas upp. Det finns en mängd olika sätt att göra detta på och även olika sätt att kunna köra båda systemen, alltså det äldre och det nya parallellt. Det finns även metoder för att kunna byta system utan att något uppehåll sker och på så sätt kunna ändra i databasen. Det svåra med detta är hur data i databasen förblir intakt och går att uppdatera. Detta ligger dock utanför detta arbetets ramar.

4 Metod

Som metod till detta arbete så användes ett experiment. Experimentet har två uppgifter, dels att skaffa förståelse för hur en migration med hjälp av det migrationsverktyg som ska användas går till, detta för att kunna skräddarsy och testa ett ramverk. Den andra delen i experimentet är att undersöka vilka problem som uppkommer när migrationen genomförs.

4.1 Experiment

Litteratur kring detta område säger att det kommer att bli problem vid migration av applikationer oberoende av vilka metoder och verktyg som används (Brodie & Stonebraker, (1993), Bisbal et. al. (1999)). Ett experiment genomfördes för att undersöka hur detta visar sig när migrationer med hjälp av Visual Studio .nets migrationsverktyg.

Experimentet genomfördes genom att först bygga upp ett antal applikationer i Visual Basic 6 och sedan migrera dessa till Visual Basic .net med hjälp av migrationsverktyget i Visual Studio .net. Applikationerna bygger, i enlighet med de avgränsningar som gjordes i problemavsnittet, mycket på kopplingar till databaser. I detta fall som även det togs upp i avgränsningen så sker kopplingen främst med hjälp av en SQLserver. Men det inkluderas även en applikation endast för att kunna testa hur gränssnittet i en applikation påverkas av migrationen. Tanken är att applikationerna som migrerats ska vara så lika de ursprungliga både i gränssnittet och i funktionaliteten. Det sker även en migration med en applikation som kopplar till en MDBdatabas då dessa går att skapa utifrån en SQLserver. Kopplingarna till SQLservern sker i två olika steg för att få så stor spridning som möjligt och därmed även fånga upp de mesta problem som kan komma att inträffa. Den första av de två är att använda textrutor för att visa och skriva in data i SQLdatabsen med hjälp av textrutornas inställningar. Den andra är att genom en datagrid och datakontroll, som finns som verktyg i Visual Studio 6, koppla till databasen. De olika applikationerna skiljs åt i både storlek och komplexitet.

Till grund för hur applikationerna kommer att utformas ligger den litteratur som finns inom området (McManus (1999), Thayer (1998), Williams (1999)). De alternativ som litteraturen ger när det kommer till koppling till databaser är främst att genom den datakontroll som finns koppla en datagrid, lista eller liknande till databasen. Ett annat alternativ som ofta förekommer är att använda sig av den datamiljö där det går att koppla till databaser genom SQLspråket. För visning av detta så finns dels textboxar som används i detta arbete och dels listor. För att koppla textrutorna till data kan dels dess egenskaper eller Visual Basic kod användas. Arbetet tar upp båda dessa alternativ.

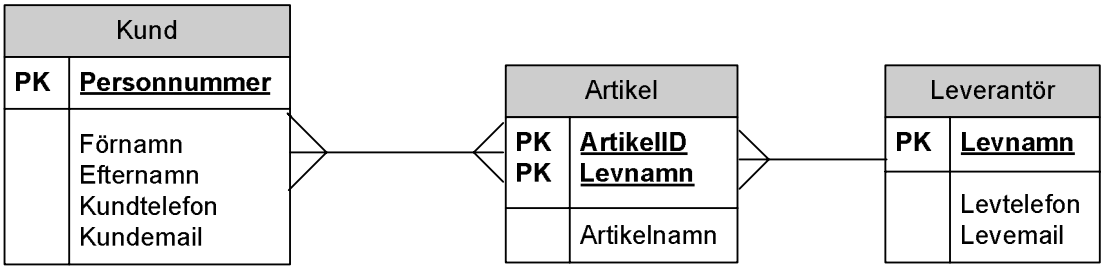
Det sker i enlighet med avgränsningarna inte någon överföring av data till olika databaser och heller inte någon kontroll huruvida det går att köra programmen parallellt under överförande fasen där själva bytet mellan de två systemen utförs.

Resultatet från experimenten är vilka problem som uppstår vid dessa specifika migrationsfall. Experimentet ger även information om hur bra verktyget migrerar koden och hur bra det är på att ge information om vad som gått fel och vad som ska göras för att rätta till problemen. Utifrån de erfarenheter som experimentet gav

skräddarsyddes ett ramverk för migration av applikationer liknande de som migrerats här och med samma verktyg.

5 Genomförande

Först så designades och byggdes en exempeldatabas upp på en lokal SQLserver. Databasen består av fyra tabeller varav en uppkommer då det finns ett många-till-många förhållande mellan tabellerna Kund och Artikel. Mellan de två tabellerna Artikel och Leverantör finns det dessutom ett en-till-många förhållande. Nedan visas en modell över hur databasen är uppbyggd.



Applikationerna byggdes upp i Visual Basic 6 med hjälp av RADmiljön Visual Studio 6. Applikationerna migrerades separat och gjordes allt mer komplexa för att bättre kunna se både små och stora fel utan att missa något i applikationerna. Det var sex stycken applikationer som migrerades. Den första har ingen koppling till någon databas utan är till för att se hur gränssnittet samt enkel Visual Basic kod migreras. Fyra utav applikationerna har koppling till den SQLdatabas som beskrivits. Det som skiljer dessa från varandra är hur kopplingen till databasen sker samt applikationernas funktionalitet. Den sista applikationen har en koppling till en .mdb databas. Denna databas är redan färdigdesignad och finns med som exempeldatabas i Visual Studio 6. En mer ingående beskrivning av applikationerna finns under resultatet samt i appendixen. För migrationen så användes det migrationsverktyg som finns i Visual Studio .net. Detta verktyg öppnar en applikation av Visual Basic 6 typ och migrerar automatiskt denna till Visual Basic. net. Verktyget genererar även en uppgraderingsrapport innehållande information om hur migrationen gått. Denna rapport ligger till grund för framställningen av de problem som senare kommer att tas upp i resultatet.

När migrationerna genomförts så skräddarsyddes ramverket utifrån de erfarenheter som experimenten givit genom att ta bort de steg som tycktes överflödiga samt lägga till steg som fattades. Detta gjordes kontinuerligt efter det att varje applikation migrerats vilket gav ett ramverk som allt mer specificerades fram ju mer erfarenheter arbetet med migrationerna gav. Beslut togs att använda ramverket Butterfly som utgångspunkt för att kunna skräddarsy ett ramverk. Anledningen att just detta ramverk valdes var på grund av att de steg som detta ramverk tog upp passade bra överens med de tänkta steg som skulle behöva genomföras för att kunna migrera. Det ansågs även vara lättast att skräddarsy ett ramverk ifrån, då stegen inte hänger ihop med varandra så mycket.

6 Resultat

Först i resultatet kommer en genomgång av alla applikationer som migrerats under genomförandet och information om hur migrationen av dessa gått. Efter detta så sker en genomgång av ändringarna i ramverket som genomförts utifrån de erfarenheter som migrationerna av applikationerna givit.

6.1 Genomgång av applikationerna

Här följer en genomgång av alla de applikationer som användes vid experimenten. Genomgången sker applikation för applikation och börjar med en kort beskrivning av vad den gör och dess gränssnitt. Sedan följer en kort utvärdering om hur migrationen gick, dels för gränssnittet och dels för applikationen.

6.1.1 Visa bild

Denna applikation har ingen koppling till någon databas vilket detta arbete inriktar sig på. Meningen med denna applikation är att kunna kontrollera hur gränssnittet samt enkel Visual Basic kod migreras. I applikationen så går det med hjälp av två radioknappar välja vilket motiv som önskas av ros eller händer (se Appendix 1). När valet genomförts trycker användaren på knappen *Visa bild* vilket triggar koden till att först kontrollera vilken bild som ska visas och sedan visa denna.

Gränssnittet:

- Uppgraderingsrapporten angav inga fel gällande designen (se Appendix 1).
- Typsnittet ändrades till Arial.

Att typsnittet ändrades till Arial medförde att rubriken innehållande texten ”Visa bild program” inte fick plats och klipptes därför av programmet. Detta kan lätt åtgärdas genom att antingen ändra storleken på typsnittet eller utöka rubrikens plats.

Applikationen:

- Uppgraderingsrapporten angav inga fel i funktionaliteten (se Appendix 1).
- Funktionaliteten i applikationen fungerade enligt de krav som ställts på den.

6.1.2 Bläddra

Applikationen kallad *Bläddra* går ut på att kunna koppla textrutor (se Appendix 2) till en SQLserver och med hjälp av två knappar kunna flytta antingen ett steg framåt eller ett steg bakåt i en tabell, i detta fall *Artikel* tabellen. Syftet med denna applikation var att kunna se om kopplingen till servern fungerade samt att se hur gränssnittet påverkades av migrationen. Kopplingen mellan servern och textrutorna gjordes med hjälp av textrutan egenskaper och inte med hjälp av Visual Basic 6 kod.

Gränssnittet:

- 9 stycken designfel i uppgraderingsrapporten (se Appendix 2).

Det blev nio stycken designfel enligt applikationens uppgraderingsrapport (se Appendix 2). Detta berodde på att egenskaperna för datamember, datasource och datafield inte uppdaterades. Detta inverkade inte på applikationens gränssnitt. Uppgraderingsrapporten gav ingen ytterligare information om vad felet kunde vara.

Det blev även ytterligare fel då gränssnittet i Visual Basic 6 använde sig av MS Sans Serif vilket migrationsverktyget ändrade till Arial. Detta är samma fel som uppstod i den första applikationen, Visa bild. I denna applikation resulterade det i att namnet på textrutan *Leverantörsnamn* delvis försvann (se Appendix 2). Detta kom inte med i uppgraderingsrapporten. För att kunna få detta att bli som i Visual Basic 6 kan antingen typsnittet ändras tillbaka eller så genomförs en ändring i etikettens storlek vilket skulle resultera i att texten kom med. Det som bör beaktas är att om ändringen i storlek sker är inte gränssnittet det samma på den äldre och den nya applikationen likvärdiga varandra.

Applikationen:

- Uppraderingsrapporten uppgav inga fel gällande funktionaliteten (se Appendix 2).
- Applikationens funktionalitet fungerade enligt de krav som ställts på den.

6.1.3 Edit

Denna applikation har samma funktion när det gäller att bläddra i tabellen och hur kopplingen till en SQLserver som applikationen *Bläddra*. Det som har lagts till här är funktionen att kunna lägga till och ta bort en förutbestämd post i tabellen *Leverantör* som används till denna applikation. Detta sker genom att två nya kommandon har lagts till som dels lägger till och dels tar bort en och samma post. De två knapparna *Lägg till* och *Ta bort* (se Appendix 3) kopplar sedan till dessa två kommandon. Någon uppdatering av data i applikationen genomförs inte förrän nästa gång den exekveras. Syftet är att se om det går att lägga till och ta bort i tabeller efter migreringen.

Gränssnittet:

- Uppgraderingsrapporten visade på 9 stycken designfel (se Appendix 3).

De nio designfelen var av samma typ som i föregående applikationer som beskrevs i avsnitt 6.2.1. varpå de inte kommer att kommenteras ytterligare här.

Applikationen:

- Uppgraderingsrapporten visade på två stycken runtimevarningar (se Appendix 3).

De två runtimevarningarna syftar på att migrationsverktyget inte klarade av att migrera alla de egenskaper som objekten *deEDIT.comDELETE* och *deEDIT.comINSERT* hade. Dessa två objekt är de som kopplar knapparna till SQLkommandona som tar bort och lägger till i tabellen. Applikationen fungerar dock som den ska även med dessa varningar. Uppgraderingsrapporten ger inte någon mer hjälp för att kunna lokalisera varför varningarna uppstod.

6.1.4 Kund

Applikationen har kvar funktionen att kunna bläddra fram och tillbaka i tabellens poster, denna gång är det tabellen *Kund* som applikationen kopplar till. I övrigt går det att lägga till en kund i tabellen. Detta genomförs genom att användaren, i fältet *Lägg till i kund* (se Appendix 4) skriver in i textrutor vad som ska lagras och sedan trycker på knappen *Lägg till*. Applikationen läser då av vad användaren skrivit in och lägger till detta på rätt plats i tabellen. Kopplingen mellan respektive textruta och plats i tabellen sker med hjälp av språket Visual Basic 6. Den nyinlagda posten i tabellen kommer direkt upp i textrutorna inom ramen *Kunder* i registret

Tabordningen i applikationen är sådan att den först går igenom de textrutor som finns i *Lägg till i kund* fältet. Efter det går den igenom knapparna för att lägga till i och navigera i tabellen och sedan knappen Stäng.

Gränssnittet:

- Uppgraderingsrapporten gav 15 designfel (se Appendix 4).

I uppgraderingsrapporten blev det 15 fel, alla av samma typ som i föregående applikationer.

Applikationen:

- Inga funtionalitetsfel rapporterades i uppgraderingsrapporten (se Appendix 4).
- Tabordningen är inte den samma som i Visual Basic 6.
- Kundens personnummer kommer inte upp i textrutan som är avsett för detta.

När applikationen körs uppstår ett fel. Personnumret på kunderna visas inte i textrutan för detta, andra data från tabellen visas på korrekt sätt. Vad detta kan bero på visar inte uppgraderingsrapporten inte heller i koden finns det uppgifter om vad som kan ha blivit fel.

Tabordningen stämmer inte överens med den satta i Visual Basic 6. Vad detta beror på ger uppgraderingsrapporten inget svar på.

6.1.5 Data grid

I denna applikation sker en koppling till en MDBdatabas vid namn *Biblio* där en mängd olika författare listas. Visningen sker i en datagrid där navigationen sker via en datakontroll. Datakontrollen kopplar databasen till datagridden och det är även via den navigationen i tabellen sker. Det går att flytta fram och tillbaka antingen ett steg eller till sista respektive första posten.

Gränssnittet:

- Uppgraderingsrapporten angav 3 designfel (se Appendix 5).

Applikationen gick inte att köra enligt de krav som ställdes på den, se nedan under applikation. Detta ledde till att designen inte kan granskas.

Applikationen:

- Uppgraderingsrapporten angav endast designfel (se Appendix 5).
- Applikationen går att köras men datakontrollen fungerar ej.

Att datakontrollen inte går att använda (se Appendix 5) gör att ingen koppling till datagridden sker. Visual Studio .net anger när applikationen körts att det inte blev några byggnadsfel trots detta så fungerar inte funktionaliteten alls. Inte heller rapporten eller de kommentarer som finns i koden (se Appendix 5) tar upp varför felet uppstår.

6.1.6 Artikel

Applikationen har tre huvudsakliga delar, en datakontroll där användaren navigerar i tabellen *Artikel*, data från denna tabell visas i tre stycken textrutor vilka är den andra delen. Den tredje delen är en datagrid där data från tabellen *Leverantör* visas.

Anledningen att data från leverantör visas är att det finns ett en-till-många förhållande mellan dessa två tabeller. Leverantören som visas i datagridden är den leverantör vars nyckel är kopplad till artikeln. Data som finns gällande *Leverantör* skall alltså uppdateras när användaren via datakontrollen bläddrar mellan posterna i tabellen *Artikel*.

Det finns tre knappar kopplade till tabellen, *Add*, *Delete* och *Update*. Dessa knappar är till för att kunna lägga till, ta bort samt ändra i tabellen. Det finns även en knapp för att kunna uppdatera databasen, *Refresh*, samt en för att stänga formuläret, *Close*.

Gränssnittet:

- I uppraderingsrapporten fanns det 8 stycken design fel (se Appendix 6).
- Felen som togs upp i uppgraderingsrapporten påverkade inte designen.

Av designfelen som framkom i uppgraderingsrapporten så var sex av dessa liknande de fel i uppgraderingen i datasource och datafield som tidigare nämnt i rapporten. De andra två var även de fel som berodde på att vissa saker inte uppdaterats i migrationen (se Appendix 6).

Applikationen:

- Uppgraderingsrapporten gav inga fel (se Appendix 6).
- Applikationen fungerade utifrån de krav som ställts på den.

6.2 Utformning av ramverket

Här följer en genomgång av de ändringar som gjorts för att skräddarsy ramverket Butterfly utifrån de riktlinjer som migrationerna och de applikationer som ska migreras bygger på.

Fas 0: Förbered migrationen: I denna fas ska stegen 0.1.1, 0.1.2 och 0.2 vara med för att dels kunna säkerställa huruvida en migration gått som det tänkt, samt bestämma hur det nya systemet ska utformas. Att fastställa användarkraven är även det viktigt, som i varje utvecklingsprojekt. Utifrån de riktlinjer och avgränsningar som detta arbete går efter är hårdvaran inte relevant vilket leder till att 0.3 inte ska vara med.

Fas 1: Förstå semantiken i det äldre systemet och utveckla det nya dataschemat: Här ligger mycket tyngd på kunskap om det äldre systemet. Även fast det anses finnas god kunskap om systemet är det alltid bra att få ner detta på papper vilket gör att steg 1.1, 1.2 och 1.3 kommer med, även 1.5 kommer med då det kan komma upp fakta under exempelvis steg 1.3 som påverkar tidigare steg. Dessa steg tar upp just förståelsen av gränssnitt, applikationen och data.

Att frysa data för att kunna använda systemen under tiden av migrationen kommer inte att ske då det i den typ av migrationsprojekt som arbetet beskriver inte behövs göras. Detta resulterade i att steg 1.6 inte är relevant. Inte heller att utveckla DAAn eller att ta fram nya datascheman och kartläggningsregler kommer att behöva göras i denna typ av projekt varpå stegen 1.3 och 1.7 tas bort.

Fas 2: Bygg upp ett exempeldatalager som, baseras på det exempeldata som tagits fram i det nya systemet: Denna fas togs bort helt för att den bygger på de data

som ska användas för att testa data samt bygga upp Chrysalisern, men byggnaden av denna är utanför arbetets gränser. Någon transformation kommer heller inte att behöva ske i det förutsatta fallet då data kommer att kunna fungera i samma tillstånd i båda miljöerna, vilket leder till att applikationerna kommer att testas på de data.

Fas 3: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen: I denna fas kommer inte några större förändringar att genomföras. Med de förutsättningar som detta arbete genomförs efter kommer migrationen av gränssnittet och applikationen i sig att göras samtidigt då det är så migrationsverktyget som används jobbar. Detta gör att stegen 3.1.1 och 3.2.1 läggs ihop till 2.1 (fas två togs ju bort helt vilket leder till att fas 3 blir till fas 2) fast testerna och samanställningen gentemot användarkraven sker separat.

Steg 3.3 där äldre komponenter ska migreras kommer inte att finnas kvar då det inte kommer att ske under den typ av migrationsprojekt som arbetet riktar sig mot. Detta medför att även steg 3.4, 3.5 samt 3.6 kommer att försvinna. Steg 3.7 kommer att vara kvar då det är av vikt att träna användarna, detta gäller dock på de applikationer som i 2.1 migrerades.

Två steg lades till där det är tänkt att de fel, dels på gränssnittet och dels på applikationens funktion, ska rättas till. Detta sker efter att migrationen slutförts och en genomgång av systemet har genomförts, dels på gränssnittsnivå och dels på applikationsnivå.

Fas 4: Migrera gradvis äldre data till det nya systemet och träna användarna i det nya systemet: Denna fas kommer inte att tas upp i arbetet då migrationen som beskrivs i arbetet sker med hjälp av migrationsverktyget som migrerar till ett språk som supportrar ADO.net från ett språk med ADO vilket medför att samma databas kan användas till båda språken.

Fas 5: Övergång till det nya systemet: Övergången kommer att ske automatiskt när migrationsverktyget har migrerat klart applikationerna, förutsatt att detta lyckas. Olika förslag på hur övergången kan ske finns under avsnittet 2.4.1.

7 Analys

Här följer en genomgång av det resultat som framkommit under de experiment som gjordes för arbetet. Analyser kommer att delas upp på dels de problem som uppstått under migrationerna och dels på analys av det ramverket som skräddarsyddes. Det finns även en tabell som kort visar vilka problem som framkom under migrationerna.

7.1 Problem vid migration

Resultatet av genomförandet visar att det kommer att bli problem vi migrering av kod vid användning av migrationsverktyget som finns inbyggd i Visual Studio .net. Resultatet visar att det kommer att uppstå både små och stora problem, inte bara på grund av de fel som uppstått under migrationen utan även efter migrationen då de fel som uppstått ska åtgärdas och inga rekommendationer anges av migrationsverktyget. Nedan följer en sammanställning av de fel som uppstått under migrationerna. Felen delas upp i tre delar, de två första är fel som uppgraderingsrapporten tagit upp, designfel och runtime varningar, medan den tredje är fel som finns på de migrerade applikationerna men som inte uppgraderingsrapporten tar upp.

Problem	**Antal applikationer**	**Allvarlighets-grad (1-3)**	**Kommentar**
Design errors			
Typsnittet uppgraderades inte	6	2	I två av applikationerna ledde detta till att information i gränssnittet uteblev.
Egenskaperna på textboxarna uppgraderades inte	4	1	Inverkade inte på varken design eller funktionalitet. Det bör beaktas att det endast var 4 applikationer som använde sig av textboxar vilket innebär att alla dessa misslyckades med migrationen av detta.
Kopplingen till adodc objektet uppgraderades inte.	1	1	Inverkade inte på varken design eller funktionalitet.
Datagridden uppgraderades inte	1	3	Funktionaliteten för dataggridden fungerade inte i applikationen.
Datakontrollen uppgraderades inte	1	3	Denna funktionalitet fungerade inte i denna applikation.
Data objektet uppgraderades inte.	1	3	Applikationen i fråga fungerade inte.
Runtime varningar			
Varning för att storleken på formuläret inte uppgraderas.	1	1	Inverkade inte på varken design eller funktionalitet.

Formuläret har nya egenskaper	1	1	Inverkade inte på varken design eller funktionalitet.
Muspekaren har nya egenskaper	1	1	Inverkade inte på varken design eller funktionalitet.
Övriga fel som inte uppgraderingsrapporten tog upp.			
Data i en textbox visades inte.	1	3	Allvarligt då kopplingen till textboxen gjordes på samma sätt som övriga i applikationen som fungerade som de skulle. Uppgraderingsrapporten gav inga indikationer på vad detta skulle kunna bero på.
Tabordningen i applikationen var inte den samma.	6	3	Detta skedde i alla applikationer, men problemet tas endast upp i en av dem då denna var till för att kontroller detta. Detta ansågs allvarligt då rapporten inte visade på att problemet uppstått.

Alla applikationerna som migrerades visade sig ha problem med att kunna överföra det typsnitt som användes i formulären. Detta kan ses som ett mindre problem som utvecklarna lätt kan åtgärda. Men när det kommer upp i större system med många formulär och därmed även mycket text så kommer arbetet med att antingen byta typsnitt eller göra ändringar i ramarna för de olika objektens storlek att ta lång tid. Om inte detta genomförs så kan information gå förlorad, både när det gäller den informationen som formuläret ger användaren men även den information som användaren får genom data som visas i formuläret. Dessutom så kan användarna känna sig främmande i det nya systemet om gränssnittet ändras för mycket.

Att de applikationer som använder sig av en koppling till en databas genom en textrutas egenskaper och de inställningar som finns för DataField, DataMember och DataSource inte uppgraderades är inte ett problem då detta beror på att Visual Basic .net sköter denna koppling på ett annat sätt som inte ska inverka på funktionaliteten och som enligt de resultat som framkommit heller inte gör detta. Det framgår även från uppgraderingsrapporterna att detta endast är ett designfel och då inte ska inverka på funktionaliteten.

Fem stycken runtimevarningar har framkommit i uppgraderingsrapporterna, två stycken i Editapplikationen och resterande tre i applikationen Artikel. Båda dessa applikationer fungerade i enlighet med de krav som ställdes på dem gällande att funktionaliteterna ska vara likvärdig de i de äldre applikationerna. Detta betraktas inte

som ett fel från migrationsverktyget då detta endast är varningar och alltså kan påverka funktionaliteterna men i dessa fall inte gör så.

De riktigt stora problemen kommer istället i applikationerna Kund samt Data Grid där funktionaliteten inte stämde överens med kraven. I Kund visades inte nyckel för en tabell i textrutan och i Data Grid så fungerade inte den datakontroll som användes i applikationen. Båda dessa hade i sina uppgraderingsrapporter endast designfel som riktade sig till gränssnittet i formulären. Migrationsverktyget gav alltså inga indikationer på vad som skulle kunna vara fel. Inte heller i de kommentarer som finns i koden för applikationerna finns det några indikationer på annat än designfel. Från detta kan det fastställas slutsatser att kopplingar till .mdb databaser är problematiska samt att det kan uppstå problem även vid koppling till en SQLdatabas.

Över det stora hela så kan det sägas att den uppgraderingsrapport som migrationsverktyget genererar visar på brister både i information om vad som gått fel och även helt brist på information om vad som ska göras åt problemen, något det är meningen att den ska göra.

7.2 Ramverket.

Genom de experiment som genomfördes gjordes ändringar för att skräddarsy ramverket Butterfly till ett ramverk för just denna typ av migrationer som framställs i arbetet. Dels så fanns det med allt för mycket om hur dataöverföring skulle ske, något som i alla migrationsfall inte är nödvändiga och som utifrån de avgränsningar som gjordes i arbetet inte behövs här varpå alla steg innehållande uppgifter om detta togs bort.

I och med att det var bestämt att migrationsverktyget skulle användas ansågs det även vara en bra idé att sätta ihop de två stegen migration av gränssnitt och migration av applikationen då dessa automatiskt sker samtidigt. På grund av det stora antal fel som uppstod vid migrationstillfällen så ansågs det även som ett naturligt beslut att inkludera ett särskilt steg där dessa fel är tänkta att rättas till.

I stort kan sägas att ramverket blivit mer specifikt för de förbestämda språken och migrationsverktygen. Detta visar sig då ramverket bygger på en uppgraderingsrapport från migrationen samt att de två språken kan hämta data från samma typ av källa. Detta gör ramverket mer användbart i praktiken då alla steg är specifika gentemot den sagda migrationen.

8 Slutsatts

Här följer en övergripande sammanfattning av resultatet som framkommit i arbetet samt en redogörelse för hur detta kan tolkas.

8.1 Resultatet

Då arbetet gått in mycket för ett visst migrationsverktyg och dessutom skräddarsytt ett ramverk för migration av applikationer med hjälp av detta verktyg så kan det lätt komma till användning i verkliga och konkreta samanhang. Resultatet som framkommit i detta arbete kan användas när en migration ska ske från Visual Basic 6 till Visual Basic .net. Arbetet kan vara till hjälp när det ska bestämmas hur en migration ska ske, samt att förbereda utvecklarna på visa problem som kan uppkomma vid migreringen. Arbetet beskriver olika metoder för hur migrationer kan genomföras samt vilka olika typer av migrationer dessa passar till. Men det är främst inom användning av det migrationsverktyg som finns i Visual Studio .net. Det som främst kan användas är det ramverk som skräddarsytts för just en sådan migration. Ramverket är tänkt att ge stöd och leda utvecklarna i migrationsprocessen, det som bör beaktas är att alla migrationsprojekt är olika. Även fast språken som ska migreras från och till är de samma så beror det, som resultatet visar, mycket på hur applikationerna är byggda, hur stora och hur komplicerade de är.

Resultatet visar att det kommer att uppstå både små och stora problem vid migration. Exempel på små problem är de designfel som finns i alla applikationer där typsnittet ändras vilket medför att information kan gå förlorad (se appendix 1-5), bland de större problemen finns det för applikationen Data Grid (se appendix 5) som inte alls fungerade samt applikationen Kund (se appendix 4) där dels tabellens nyckel inte visades i dess textruta och inte heller tabordningen var den samma. Utvecklare måste därför förbereda sig på att detta kommer att ske. Resultatet visar även att den rapport som migrationsverktyget genererar inte tar upp alla fel som inverkar på funktionalitet och gränssnitt och inte heller beskriver hur felen ska rättas till, även här kan applikationen Data Grid användas som exempel där endast tre designfel tas upp av rapporten men där applikationen inte alls fungerar som den ska (se appendix 5). Varken vad som kan vara fel eller vad som kan göras åt felet tas här upp av rapporten. Även detta måste utvecklare vara införstådda med, både när det gäller att använda verktyget men även vid val huruvida det ska användas överhuvudtaget.

I stort så stämmer arbetet överens med vad som tidigare litteratur har sagt. Många tar upp att det kommer att bli problem med migration, de tar även upp att det beror mycket på hur applikationen som migreras är uppbyggd. Litteraturen tar även upp att olika ramverk passar mer eller mindre bra till olika typer av migrationsprojekt. Det detta arbete tagit upp mer än andra är att gå in mer och undersöka hur migration med ett förutbestämt migrationsverktyg på bästa sätt kan ske och vilka problem som framkommer under en sådan process. Andra kan ha användning av detta arbete när de ska ta fram ramverk för migration med liknande migrationsverktyg.

9 Diskussion

Här följer en diskussion om arbetet utifrån resultatet och den analys som genomfördes på detta. Det kommer att tas upp vad som är bra och dåligt i resultatet och i arbetet, även metoden som användes till att utföra arbetet kommer att tas upp och diskuteras. Sist i kapitlet så kommer en rad förslag på fortsatta arbeten som bygger på denna rapport att tas upp.

9.1 Resultatet

Resultatet visar att det kommer att ske problem vid användande av det migrationsverktyg som Microsoft har utvecklat för Visual studio .net för att kunna migrera applikationer från Visual Basic 6 till det nya .net. Detta blir till ett problem då Microsoft kommer att vilja att sina kunder bytet till Visual Basic .net. Ramverket som skräddarsyddes för migreringen fungerade bra. Att kontinuerligt uppdatera ramverket efter varje migration och sedan även använda sig av det vid nästa migration gör att det går att säga att metoden kunde visa att ramverket i praktiskt utnyttjande fungerade bra.

De exempel på problem som inträffat under arbetets gång har varit både lika varandra, exempelvis att typsnitten ändrades, men det har även inträffat en hel det olika problem som att datakontrollen inte fungerade (se appendix 5) och att inte nyckeln i en applikation visades korrekt (se appendix 4). Detta visar på att det kommer att bli problem men det visar även på att det är svårt att förutse problemen då det i exemplet med att nyckeln inte visades ändvändes samma metod som i andra applikationer där detta problem inte uppstod under migrationen. Även fast en rad olika problem uppkommit under rapporten så kan inte det ses som att detta är alla problem som kan uppstå. Men att få med alla fel i arbetet skulle vara en omöjlighet då det beror så mycket på hur applikationen är uppbyggd och att uppbyggnaden kan variera i så otroligt många olika sätt. Detta gör att vissa av problemen som inträffat skulle kunna ses som tillfälligheter men visa problem kommer med säkerhet att inträffa, vilket medför att arbetet mer bör ses som exempel på hur det kan gå och inte exempel på hur det kommer att gå. Men det står ändå klart att problem kommer att ske i på ett eller annat sätt. Det ramverk som framtagits skulle dock vara lika uppbyggd oberoende av vilka problem som uppkommer.

Det framgår från resultatet att det är beroende på vilket typ av applikation som migreras som olika typer av problem uppkommer. Om ett större antal applikationer med ett vidare användningsområde skulle ha migrerats så skulle en kartläggning av problem som uppstår vid olika typer av applikationer kunna ha genomförts. Ett större antal applikationer skulle även ge säkrare fakta gällande vad som går att migrera och vad som inte går att migrera, men detta skulle kräva mer resurser än vad som tilldelats detta arbete. Det är heller inte det som arbetet riktar sig mot.

Att ramverket riktar sig specifikt till en migration mellan språken Visual Basic 6 och Visual Basic .net med hjälp av Visual Studio .nets migrationsverktyg är positivt då tanken bakom detta är att de ramverk som idag finns på marknaden inte är specifika nog och kan därför medföra extra arbete vid användning av dessa. Att ramverket är specifikt medför att utvecklaren inte känner sig så vilsen i ramverket då stegen i ramverket stämmer med de steg som migrationsvertyget utför. Ramverket ger på detta

sätt även mer stöd åt användaren i vad denne ska göra. Detta gör att ramverket blir med användbart i en praktisk situation.

Inte bara hur specifikt ramverket går in på att det är just de bestämda språken och verktyget spelar in på hur detaljerat det ska vara utan även hur specifikt ramverket ska vara på varje punkt. Ju mer detaljerat ramverket är desto bättre stöd får användaren. Det som skulle kunna göras är att i de olika stegen i ramverket beskriva och ge förslag på hur det specifika steget ska genomföras.

9.2 Metoden

Metoden experiment fungerar bra för detta arbete då det i tidigare litteratur redan framkommit att det kommer att bli problem vid migration av kod, vilket resulterat i att det redan fanns en bild av vad som skulle komma att hända vid de olika migrationstillfällena. Metoden hjälper till att dels visa på att problem faktiskt uppkommer samt vad för problem som uppkommer och vilka som är vanligast vid just den typ av migration som arbetet bygger på. Metoden ger även erfarenheter för att kunna skräddarsy det ramverk som presenteras i rapporten.

Den kritik som kan anknytas till metoden är att den främst är till för att bevisa hypoteser, vilket den också gör, men att den inte är utvecklad för att visa på att en ny metod som arbetet tar fram verkligen stämmer. Det är i detta arbete det skräddarsydda ramverket som måste testas för att se huruvida det är användbart i verkligheten. Men det sätt som metoden användes på gör att den har mycket av en implementation i sig, vilken är den metod som främst använda för att bevisa något som framkommit.

9.3 Framtida arbeten

Ett naturligt fortsatt arbete vore att fortsätta med att undersöka vilka problem som uppstår vid migrationer med hjälp av samma språk och verktyg men att inte inrikta sig på applikationer kopplade till en SQLdatabas. Applikationerna skulle exempelvis kunna rikta sig mer mot mer avancerad programmering i Visual Basic 6 eller andra typer av kopplingar till en databas än via en SQLserver. Det skulle då kunna gå att jämföra resultaten från de migrationerna med resultatet i denna rapport och få ut än mer utförligare analyser.

En annan aspekt inom migrationen som inte tas upp i detta arbete är val av språk som applikationerna ska migreras till, bara inom .net miljön så finns det än mängd olika språk som kan tänkas användas. Är det bättre att använda sig av exempelvis C# att migrera till då detta språken är kraftigare än Visual Basic .net? Då uppstår en annan fråga, hur ska migrationen till detta språk, eller något annat, genomföras? Ska ett verktyg användas även till denna migration eller är det i detta fall bättre att inte använda sig av något migrationsverktyg? Hur väl fungerar dessa migrationsverktyg i jämförelse med det som finns i Visual Studio .net?

När frågan om migration till andra språk med hjälp av andra migrationsverktyg kommer upp så kommer även förslaget om att skräddarsy ramverk även till andra typer av migrering. Migrera utan något verktyg, från och till olika språk. Migrering med hjälp av andra verktyg som finns på marknaden. Det skulle även kunna skräddarsys ett ramverk för hur man ska behandla data under migrationsfasen, något som detta arbete inte tar upp. Ska det utföras med hjälp av Gateways som i Chicken little eller med en DAA och Chrysaliser som i Butterfly. En annan frågeställning som

kan göras är att mer konkret gå in på hur cut-over fasen ska genomföras och vad det finns för nackdelar och fördelar med de olika metoderna.

En frågeställning som kommer upp från detta arbete är hur de problem som blir vid migrationen med hjälp av det migrationsverktyget som används i detta arbete, ska lösas. Går det att göra en mer tydlig mall att gå efter eller är problemen så olika varandra och så oförutsägbara att det vore en omöjlighet att sätta upp ett bestämt sätt att lösa problemen på? Det skulle ändå kunna gå att ge förslag på hur den information som migrationsverktyget ger i den uppgraderingsrapport som genereras efter varje migration ska tydas och vad för hjälp den kan ge, om någon hjälp alls.

Referenser

Abramson, D. & Watson, G. (2003) *Debugging scientific applications in the .NET Framework*. Monash University, Australien. Tillgänglig på Internet: http://www.csse.monash.edu.au/~davida/papers/VSGuard.pdf [Hämtad 05.03.17]

Berntsson, M., Hansson, J., Olsson, B. & Lundell, B. (2002) *Planning and implementing your final year project with success! A guide for students in computer science and information systems*. Storbrittanien: Springer-Verlag London Limited.

Bisbal, J., Lawless, D., Wu, B. & Grimson. J. (1999) Legacy information systems: Issues and directions. Trinity College Dublin, Irland. Tillgänglig på Internet: www.comp.dit.ie/bwu/IEE ESoftware16_5.pdf [Hämtad 05.02.24]

Brodie, M. & Stonebraker, M. (1993) DARWIN: On the incremental migration of legacy information systems. College of engineering, University of Califonia, USA. Tillgänglig på Internet: http://db.cs.berkeley.edu//papers/S2K-93-25.pdf [Hämtad 05.03.21]

Cantù, M., (2003) *Mastering Delphi 7.* USA: SYBEX, Inc.

Dietel, H., Dietel, P. & Nieto, T. (2002) *Visual Basic .net how to program, second edition.* USA: Prentice-Hall, Inc.

Dobson, R. (2001) *Programming Microsoft Access version 2002*. USA: Microsoft Press.

Fedorov, A. (2002) *A programmer's guide to .NET.* Great Britain: Biddles Ltd of Guildford and King's Lynn.

Heasman, J. (2004) Migrating to the .NET platform: an introduction. Storbrittanien: NGS Software. Tillgänglig på Internet: http://www.compseconline.com/free_articles/nese_apr04.pdf [Hämtad 05.03.17]

Jonson, T. (2002) *Visual Basic i focus – Grundläggande applikationsutvecklig och programmering version 6.* Lund: Studentlitteratur.

Jonson, T. (2003) *Programering i focus – Grundläggande programutveckling i Visual Basic .NET.* Lund: Studentlitteratur.

Mackenzie, D. & Sharky, K. (2002) *Lär dig Visual Basic .NET på 3 veckor*. Falun: ScandBook AB.

McManus, J. (1999) *Database Access with Visual Basic 6*. USA: Sams Publishing.

Mössenböck, H., Beer, W., Birngruber, D. & Wöβ, A. (2003) *.NET application development with C#, ADO.NET, ASP.NET and Web Services*. Great Britain: Henry Ling Ltd.

Perry, G. (1999) *Lär dig Visual Basic 6 på 3 veckor*. Falun: Ait.

Price, J. (2003) *C# database programming*. USA: SYBEX, Inc.

Sainio, A. (2000) *Access 2000*. Finland: WS Bookwell

Thayer, R. (1998) *Visual Basic 6 Unleashed*. USA, Sams publishing.

Williams, R. (1999) *Professional Visual Basic 6 databases*. USA: Wrox Press.

Wu, B., Lawless, D., Bisbal, J., Ricardson, R., Grimson, J., Wade, V. & O´sullivan, D. (1997) The Butterfly Methodology: A Gateway-free Approach for Migrating Legacy Information Systems. *Presenterad vid The 3rd IEEEE Conference on Engeneering of Complex Computer Systems*, Como, Italien September 8-12, 1997.

Appendix 1: Visa bild

Gränssnittet.

(Bild 1: Bläddra i Visual Basic 6)

(Bild 2: Bläddra i Visual Basic .net)

Koden.

Visual Basic 6:

```
Private Sub Command1_Click()
```

```
If optROS = True Then
Image1.Picture = LoadPicture("D:\Bilder\DM\ros.gif")
End If

If optHÄNDER = True Then
Image1.Picture = LoadPicture("D:\Bilder\DM\hands.jpg")
End If

End Sub
```

Visual Basic .net:

(Här visas hela koden från Visual Basic .net i resterande Appendix kommer endast den koden som reffererar till den ursprungliga koden.)

```
Option Strict Off
Option Explicit On
Friend Class formVisabild
	Inherits System.Windows.Forms.Form
#Region "Windows Form Designer generated code "
	Public Sub New()
		MyBase.New()
		If m_vb6FormDefInstance Is Nothing Then
			If m_InitializingDefInstance Then
				m_vb6FormDefInstance = Me
			Else
				Try
					'For the start-up form, the first instance created is the default instance.
					If System.Reflection.Assembly.GetExecutingAssembly.EntryPoint.DeclaringType Is Me.GetType Then
						m_vb6FormDefInstance = Me
					End If
				Catch
				End Try
			End If
		End If
		'This call is required by the Windows Form Designer.
		InitializeComponent()
	End Sub
	'Form overrides dispose to clean up the component list.
```

```
      Protected Overloads Overrides Sub Dispose(ByVal Disposing As
Boolean)
            If Disposing Then
                  If Not components Is Nothing Then
                        components.Dispose()
                  End If
            End If
            MyBase.Dispose(Disposing)
      End Sub
      'Required by the Windows Form Designer
      Private components As System.ComponentModel.IContainer
      Public ToolTip1 As System.Windows.Forms.ToolTip
      Public WithEvents optHÄNDER As System.Windows.Forms.RadioButton
      Public WithEvents optROS As System.Windows.Forms.RadioButton
      Public WithEvents Command1 As System.Windows.Forms.Button
      Public WithEvents Image1 As System.Windows.Forms.PictureBox
      Public WithEvents Label1 As System.Windows.Forms.Label
      'NOTE: The following procedure is required by the Windows Form
Designer
      'It can be modified using the Windows Form Designer.
      'Do not modify it using the code editor.
      <System.Diagnostics.DebuggerStepThrough()>      Private      Sub
InitializeComponent()
            Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(formVisabild))
            Me.components = New System.ComponentModel.Container()
            Me.ToolTip1                        =                        New
System.Windows.Forms.ToolTip(components)
            Me.ToolTip1.Active = True
            Me.optHÄNDER = New System.Windows.Forms.RadioButton
            Me.optROS = New System.Windows.Forms.RadioButton
            Me.Command1 = New System.Windows.Forms.Button
            Me.Image1 = New System.Windows.Forms.PictureBox
            Me.Label1 = New System.Windows.Forms.Label
            Me.BackColor                                                =
System.Drawing.SystemColors.InactiveCaption
            Me.Text = "Visa Bild Program"
            Me.ClientSize = New System.Drawing.Size(469, 285)
            Me.Location = New System.Drawing.Point(4, 30)
            Me.StartPosition                                            =
System.Windows.Forms.FormStartPosition.WindowsDefaultLocation
            Me.Font    =    New    System.Drawing.Font("Arial",    8!,
System.Drawing.FontStyle.Regular,  System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
```

```
            Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
            Me.FormBorderStyle                                          =
System.Windows.Forms.FormBorderStyle.Sizable
            Me.ControlBox = True
            Me.Enabled = True
            Me.KeyPreview = False
            Me.MaximizeBox = True
            Me.MinimizeBox = True
            Me.Cursor = System.Windows.Forms.Cursors.Default
            Me.RightToLeft = System.Windows.Forms.RightToLeft.No
            Me.ShowInTaskbar = True
            Me.HelpButton = False
            Me.WindowState                                              =
System.Windows.Forms.FormWindowState.Normal
            Me.Name = "formVisabild"
            Me.optHÄNDER.TextAlign                                      =
System.Drawing.ContentAlignment.MiddleLeft
            Me.optHÄNDER.Text = "Händer"
            Me.optHÄNDER.CausesValidation = False
            Me.optHÄNDER.Size = New System.Drawing.Size(113, 17)
            Me.optHÄNDER.Location  =  New  System.Drawing.Point(328,
128)
            Me.optHÄNDER.TabIndex = 3
            Me.optHÄNDER.Font = New System.Drawing.Font("Arial", 8!,
System.Drawing.FontStyle.Regular,  System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.optHÄNDER.CheckAlign                                     =
System.Drawing.ContentAlignment.MiddleLeft
            Me.optHÄNDER.BackColor                                      =
System.Drawing.SystemColors.Control
            Me.optHÄNDER.Enabled = True
            Me.optHÄNDER.ForeColor                                      =
System.Drawing.SystemColors.ControlText
            Me.optHÄNDER.Cursor                                         =
System.Windows.Forms.Cursors.Default
            Me.optHÄNDER.RightToLeft                                    =
System.Windows.Forms.RightToLeft.No
            Me.optHÄNDER.Appearance                                     =
System.Windows.Forms.Appearance.Normal
            Me.optHÄNDER.TabStop = True
            Me.optHÄNDER.Checked = False
            Me.optHÄNDER.Visible = True
            Me.optHÄNDER.Name = "optHÄNDER"
            Me.optROS.TextAlign                                         =
System.Drawing.ContentAlignment.MiddleLeft
```

```
            Me.optROS.Text = "Ros"
            Me.optROS.CausesValidation = False
            Me.optROS.Size = New System.Drawing.Size(113, 17)
            Me.optROS.Location = New System.Drawing.Point(328, 104)
            Me.optROS.TabIndex = 2
            Me.optROS.Font = New System.Drawing.Font("Arial", 8!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.optROS.CheckAlign =
System.Drawing.ContentAlignment.MiddleLeft
            Me.optROS.BackColor = System.Drawing.SystemColors.Control
            Me.optROS.Enabled = True
            Me.optROS.ForeColor =
System.Drawing.SystemColors.ControlText
            Me.optROS.Cursor = System.Windows.Forms.Cursors.Default
            Me.optROS.RightToLeft =
System.Windows.Forms.RightToLeft.No
            Me.optROS.Appearance =
System.Windows.Forms.Appearance.Normal
            Me.optROS.TabStop = True
            Me.optROS.Checked = False
            Me.optROS.Visible = True
            Me.optROS.Name = "optROS"
            Me.Command1.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
            Me.Command1.BackColor =
System.Drawing.SystemColors.Control
            Me.Command1.Text = "Visa bild"
            Me.Command1.Size = New System.Drawing.Size(57, 25)
            Me.Command1.Location = New System.Drawing.Point(352, 160)
            Me.Command1.TabIndex = 1
            Me.Command1.Font = New System.Drawing.Font("Arial", 8!,
System.Drawing.FontStyle.Regular, System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.Command1.CausesValidation = True
            Me.Command1.Enabled = True
            Me.Command1.ForeColor =
System.Drawing.SystemColors.ControlText
            Me.Command1.Cursor = System.Windows.Forms.Cursors.Default
            Me.Command1.RightToLeft =
System.Windows.Forms.RightToLeft.No
            Me.Command1.TabStop = True
            Me.Command1.Name = "Command1"
            Me.Image1.Size = New System.Drawing.Size(161, 177)
            Me.Image1.Location = New System.Drawing.Point(96, 64)
```

```
            Me.Image1.Enabled = True
            Me.Image1.Cursor = System.Windows.Forms.Cursors.Default
            Me.Image1.SizeMode                                      =
System.Windows.Forms.PictureBoxSizeMode.Normal
            Me.Image1.Visible = True
            Me.Image1.BorderStyle                                   =
System.Windows.Forms.BorderStyle.None
            Me.Image1.Name = "Image1"
            Me.Label1.BackColor                                     =
System.Drawing.SystemColors.InactiveCaption
            Me.Label1.Text = "Visa bild program"
            Me.Label1.Font  =  New  System.Drawing.Font("Arial",  18!,
System.Drawing.FontStyle.Regular,  System.Drawing.GraphicsUnit.Point,
CType(0, Byte))
            Me.Label1.Size = New System.Drawing.Size(193, 33)
            Me.Label1.Location = New System.Drawing.Point(159, 16)
            Me.Label1.TabIndex = 0
            Me.Label1.TextAlign                                     =
System.Drawing.ContentAlignment.TopLeft
            Me.Label1.Enabled = True
            Me.Label1.ForeColor                                     =
System.Drawing.SystemColors.ControlText
            Me.Label1.Cursor = System.Windows.Forms.Cursors.Default
            Me.Label1.RightToLeft                                   =
System.Windows.Forms.RightToLeft.No
            Me.Label1.UseMnemonic = True
            Me.Label1.Visible = True
            Me.Label1.AutoSize = False
            Me.Label1.BorderStyle                                   =
System.Windows.Forms.BorderStyle.None
            Me.Label1.Name = "Label1"
            Me.Controls.Add(optHÄNDER)
            Me.Controls.Add(optROS)
            Me.Controls.Add(Command1)
            Me.Controls.Add(Image1)
            Me.Controls.Add(Label1)
      End Sub
#End Region
#Region "Upgrade Support "
      Private Shared m_vb6FormDefInstance As formVisabild
      Private Shared m_InitializingDefInstance As Boolean
      Public Shared Property DefInstance() As formVisabild
            Get
```

```
                  If    m_vb6FormDefInstance    Is    Nothing    OrElse
m_vb6FormDefInstance.IsDisposed Then
                        m_InitializingDefInstance = True
                        m_vb6FormDefInstance = New formVisabild()
                        m_InitializingDefInstance = False
                  End If
                  DefInstance = m_vb6FormDefInstance
            End Get
            Set
                  m_vb6FormDefInstance = Value
            End Set
      End Property
#End Region
      Private Sub Timer1_Timer()

      End Sub

      Private Sub Command1_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles Command1.Click

            If optROS.Checked = True Then
                  Image1.Image                                        =
System.Drawing.Image.FromFile("D:\Bilder\DM\ros.gif")
            End If

            If optHÄNDER.Checked = True Then
                  Image1.Image                                        =
System.Drawing.Image.FromFile("D:\Bilder\DM\hands.jpg")
            End If

      End Sub
End Class
```

Uppgraderingsrapporten från Visual Basic .net.

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
(Global Issues)				0	0	0

Global update issues:
None

visabildprogram.vb	visabildprogram.frm	Form	Upgraded	0	0	0
Upgrade Issues for visabildprogram.frm:						
None						
1 File(s)		Forms: 1	Upgraded: 1 Not upgraded: 0	0	0	0

Click here for help with troubleshooting upgraded projects

Upgrade Settings

LogFile: visabildprogram.log

MigrateProjectTo: WinExe

OutputDir: D:\Skolan\Xjobb\VB.net program\VISABILD

OutputName: visabildprogram.vbproj

ProjectName: visabildprogram

ProjectPath: D:\Skolan\Xjobb\VB6program\VISABILD\visabildprogram.vbp

Genomgång av ramverket.

Fas 0: Förbered migrationen

0.1 Fastställ krav för mätning av hur bra migrationen gått.

Migrationsprocessen ska, med hjälp av den wizard som finns tillgänglig i Visual Studio.net, problemfritt kunna migrera gränssnittet samt applikationen till det nya Visual Basic .net. Detta ska ske utan att några ytterligare modifikationer ska behöva göras för att upprätthålla gränssnittet eller funktionaliteten i applikationen.

0.2 Fastställ hur det nya systemet ska byggas upp

Systemet ska bygga på Visual Basic 6 kod.

Fas 1: Förstå semantiken i det äldre systemet

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.

Applikationen har en rubrik innehållande texten ”Visa Bild Program” överst i formuläret. Under denna till höger finns det två radioknappar med namnen Ros och Händer. Under desa knapparna finns en knapp som heter Visa bild. Till höger om knapparna kommer Bilderna att komma upp.

Några överflödigheter kunde inte identifieras.

Det nya gränssnittet ska vara likvärdigt med det äldre när det gäller färg, form, placering samt typsnitt.

1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

Användaren kan med hjälp av de två radioknapparna som finns i applikationen välja vilken bild som ska visas. Bilden visas genom att trycka på knappen Visa bild. Detta sker med hjälp av Visual Basic kod.

Några överflödigheter kunde inte identifieras.

Den nya applikationen ska på ett likvärdigt sätt kunna utföra de funktioner som den äldre applikationen kunde göra.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ funktionerna för de nya data.

Ingen yttre data används till denna applikation.

1.5 Slutför migrationskraven

Inga ytterligare migrationskrav kunde identifieras.

Fas 2: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen

2.1 Migrera det äldre gränssnittet och applikationen

Migrationen genomfördes med hjälp av den wizard som finns inbyggd i Visual Studio .net.

2.1.1 Testa gränssnittet mot data

Tester av gränssnittet utfördes.

2.1.2 Bekräfta gentemot kraven på gränssnittet

Typsnittet som användes hade ändrats vilket medförde att delar av rubriken inte kom med. I övrigt stämde gränssnittet gentemot kraven.

2.1.3 Rätta till fel som uppstått i gränssnittet

Felen rättades till genom att utöka rubrikens ram.

2.1.4 Testa applikationen mot data

Tester av applikationen utfördes.

2.1.5 Bekräfta gentemot kraven på applikationen

Applikationen fungerade som den skulle utifrån de krav som ställts på den.

2.1.6 Rätta till fel som uppstått i applikationen

Inga fel hade uppstått.

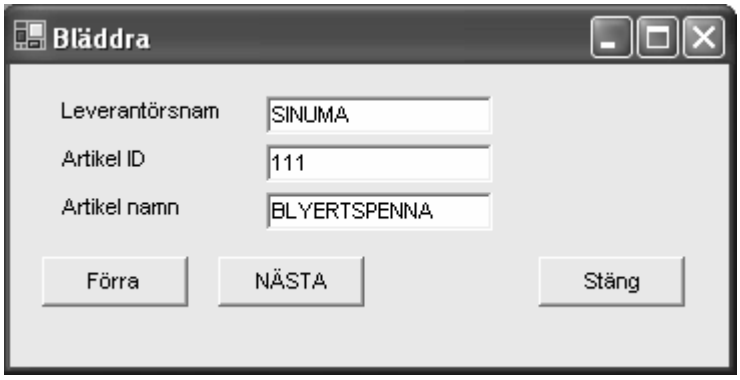
2.2 Träna användarna på de nya applikationerna

Inga användare finns till denna applikation varpå ingen träning genomfördes.

Appendix 2: Bläddra

Gränssnittet.

(Bild 1: Bläddra i Visual Basic 6)



(Bild 2: Bläddra i Visual Basic .net)

Koden.

Visual Basic 6:

```
Private Sub comFÖRRA_Click()
deBLÄDDRA.rscomBLÄDDRA.MovePrevious
End Sub

Private Sub comNÄSTA_Click()
deBLÄDDRA.rscomBLÄDDRA.MoveNext
End Sub

Private Sub comSTÄNG_Click()
End
End Sub
```

Visual Basic.net:

```
Option Strict Off
Option Explicit On
      Private Sub comFÖRRA_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles comFÖRRA.Click

            deBLÄDDRA.rscomBLÄDDRA.MovePrevious()

      End Sub

      Private Sub comNÄSTA_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles comNÄSTA.Click

            deBLÄDDRA.rscomBLÄDDRA.MoveNext()

      End Sub

      Private Sub comSTÄNG_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles comSTÄNG.Click

            End

      End Sub
      Public Sub VB6_AddADODataBinding()
            deBLÄDDRA.comBLÄDDRA()
            ADOBind_comBLÄDDRA = New VB6.MBindingCollection()
            ADOBind_comBLÄDDRA.DataSource = deBLÄDDRA
            ADOBind_comBLÄDDRA.DataMember = "comBLÄDDRA"
            ADOBind_comBLÄDDRA.Add(Text4,  "Text",  "ARTID",  Nothing,
"Text4")
            ADOBind_comBLÄDDRA.Add(Text3, "Text", "ARTNAMN", Nothing,
"Text3")
            ADOBind_comBLÄDDRA.Add(Text1, "Text", "LEVNAMN", Nothing,
"Text1")
            ADOBind_comBLÄDDRA.UpdateMode                              =
VB6.UpdateMode.vbUpdateWhenPropertyChanges
            ADOBind_comBLÄDDRA.UpdateControls()
      End Sub
      Public Sub VB6_RemoveADODataBinding()
            ADOBind_comBLÄDDRA.Clear()
            ADOBind_comBLÄDDRA.Dispose()
            ADOBind_comBLÄDDRA = Nothing
      End Sub
```

```
    Private Sub labLEVNAMN_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles labLEVNAMN.Click


    End Sub
End Class
```

Uppgraderingsrapporten från Visual Basic .net.

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
(Global Issues)				0	0	0
Global update issues:						
None						
deBLÄDDRA.vb	deBLÄDDRA.Dsr	ActiveX Designer	Upgraded	0	0	0
Upgrade Issues for deBLÄDDRA.Dsr:						
None						
formBLÄDDRA.vb	formBLÄDDRA.frm	Form	Upgraded with issues	9	0	9
Upgrade Issues for formBLÄDDRA.frm:						

#	Severity	Location	Object Type	Object Name	Property	Description
1	Design Error	(Layout)	TextBox	Text1	DataField	TextBox property Text1.DataField was not upgraded.
2	Design Error	(Layout)	TextBox	Text1	DataMember	TextBox property Text1.DataMember was not upgraded.
3	Design Error	(Layout)	TextBox	Text1	DataSource	TextBox property Text1.DataSource was not upgraded.
4	Design Error	(Layout)	TextBox	Text3	DataField	TextBox property Text3.DataField was not upgraded.
5	Design Error	(Layout)	TextBox	Text3	DataMember	TextBox property Text3.DataMember

						was not upgraded.
6	Design Error	(Layout)	TextBox	Text3	DataSource	TextBox property Text3.DataSource was not upgraded.
7	Design Error	(Layout)	TextBox	Text4	DataField	TextBox property Text4.DataField was not upgraded.
8	Design Error	(Layout)	TextBox	Text4	DataMember	TextBox property Text4.DataMember was not upgraded.
9	Design Error	(Layout)	TextBox	Text4	DataSource	TextBox property Text4.DataSource was not upgraded.

2 File(s)	ActiveX Designers: 1 Forms: 1	Upgraded: 9 2 Not upgraded: 0	0	9	

Click here for help with troubleshooting upgraded projects

Upgrade Settings

LogFile: proBLÄDDRA.log

MigrateProjectTo: WinExe

OutputDir: D:\Skolan\Xjobb\VB.net program\BLÄDDRA

OutputName: proBLÄDDRA.vbproj

ProjectName: proBLÄDDRA

ProjectPath: D:\Skolan\Xjobb\VB6program\BLÄDDRA\proBLÄDDRA.vbp

Genomgång av ramverket.

Fas 0: Förbered migrationen

0.1 Fastställ krav för mätning av hur bra migrationen gått.

Migrationsprocessen ska, med hjälp av den wizard som finns tillgänglig i Visual Studio.net, problemfritt kunna migrera gränssnittet samt applikationen till det nya Visual Basic .net. Detta ska ske utan att några ytterligare modifikationer ska behöva göras för att upprätthålla gränssnittet eller funktionaliteten i applikationen.

0.2 Fastställ hur det nya systemet ska byggas upp

Uppbyggnaden ska ske på samma sätt som i det äldre, en koppling till en databas ska ske genom en OLDBprovider för en SQLserver.

Det ska sedan gå att bläddra i en tabell med hjälp av två knappar och textrutor.

Fas 1: Förstå semantiken i det äldre systemet

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.

Gränssnittet har textsträngar före alla textrutor som talar om vad som kommer att visas i dessa. Efter varje textsträng så finns en textruta. Under textsträngarna och textrutorna befinner sig två knappar som man manövrerar fram och tillbaka i tabellen. Längs ner till höger finns en knapp som stänger formuläret.

Några överflödigheter kan inte identifieras

Funktionaliteten i den nya gränssnittet ska se lika ut som i det äldre vad gäller storlek, placering, färg och typsnitt.

1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

Applikationen kopplar genom en OLDBprovider till SQLservern och kopplar alla textrutor och knapparna Förra och Nästa till tabellen Artikel i Northvind databasen. Det går genom att flytta sig fram och tillbaka i tabellens poster med hjälp av de två knapparna Förra och Nästa. Knappen Stäng stänger ner hela formuläret.

Några överflödigheter kan inte identifieras.

Funktionalitet i den nya applikationen ska fungera på samma sätt som i den äldre applikationen.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ funktionerna för de nya data.

Data består av tabellen Artikel och består av tre kolumner, Levnamn, ArtID och Artnamn. Tabellen består av fem poster. Levnamn kommer från tabellen Leverantör som har en koppling till Artikel.

Inga överflödigheter kunde identifieras.

Funktionerna för den nya datan ska vara likvärdig den i det äldre.

1.5 Slutför migrationskraven

Inga ytterligare migrationskrav kunde identifieras.

Fas 2: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen

2.1 Migrera det äldre gränssnittet och applikationen

Migrationen genomfördes med hjälp av den wizard som finns inbyggd i Visual Studio .net.

2.1.1 Testa gränssnittet mot data

Tester av gränssnittet utfördes.

2.1.2 Bekräfta gentemot kraven på gränssnittet

Typsnittet blev inte det samma, i det äldre gränssnittet användas MS Sans Serif men i det nya så var det Arial som användes. Detta bekräftades av den uppgraderingsrapporten som säger att det finns 9 design erors (se appendix B1).

2.1.3 Rätta till fel som uppstått i gränssnittet

Felen rättades till genom att utöka rubrikens ram.

2.1.4 Testa applikationen mot data

Tester av applikationen utfördes.

2.1.5 Bekräfta gentemot kraven på applikationen

Funktionaliteten var likvärdig den i den äldre applikationen.

2.1.6 Rätta till fel som uppstått i applikationen

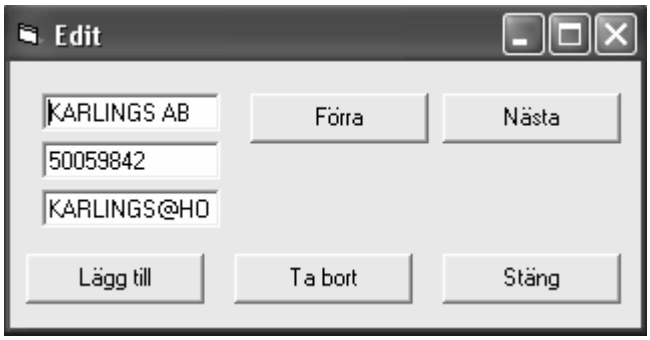
Inga fel hade uppstått.

2.2 Träna användarna på de nya applikationerna

Inga användare finns till denna applikation varpå ingen träning genomfördes.

Appendix 3: Edit

Gränssnittet.



(Bild 1: Edit i Visual Basic 6)

(Bild 2: Edit i Visual Basic .net)

Koden.

Visual Basic 6:

```
Private Sub butFÖRRA_Click()
deEDIT.rscomEDIT.MovePrevious
End Sub

Private Sub butLÄGGTILL_Click()
deEDIT.comINSERT
End Sub

Private Sub butNÄSTA_Click()
deEDIT.rscomEDIT.MoveNext
End Sub

Private Sub butSTÄNG_Click()
```

End

End Sub

Private Sub butTABORT_Click()

deEDIT.comDELETE

End Sub

Visual Basic .net:

```
Option Strict Off

Option Explicit On

      Private Sub butFÖRRA_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles butFÖRRA.Click


            deEDIT.rscomEDIT.MovePrevious()


      End Sub


      Private    Sub    butLÄGGTILL_Click(ByVal    eventSender    As
System.Object,   ByVal   eventArgs   As   System.EventArgs)   Handles
butLÄGGTILL.Click


            'UPGRADE_WARNING: Couldn't resolve default property of
object      deEDIT.comINSERT.      Click      for      more:      'ms-
help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'

        deEDIT.comINSERT()


      End Sub


      Private Sub butNÄSTA_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles butNÄSTA.Click


            deEDIT.rscomEDIT.MoveNext()


      End Sub


      Private Sub butSTÄNG_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles butSTÄNG.Click


            End


      End Sub
```

```
	Private Sub butTABORT_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles butTABORT.Click


		'UPGRADE_WARNING: Couldn't resolve default property of
object	deEDIT.comDELETE.	Click	for	more:	'ms-
help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup1037"'
		deEDIT.comDELETE()


	End Sub
	Public Sub VB6_AddADODataBinding()
		deEDIT.comEDIT()
		ADOBind_comEDIT = New VB6.MBindingCollection()
		ADOBind_comEDIT.DataSource = deEDIT
		ADOBind_comEDIT.DataMember = "comEDIT"
		ADOBind_comEDIT.Add(Text3, "Text", "LEVTELEFON", Nothing,
"Text3")
		ADOBind_comEDIT.Add(txtEMAIL,	"Text",	"LEVEMAIL",
Nothing, "txtEMAIL")
		ADOBind_comEDIT.Add(txtNAMN, "Text", "LEVNAMN", Nothing,
"txtNAMN")
		ADOBind_comEDIT.UpdateMode
=
VB6.UpdateMode.vbUpdateWhenPropertyChanges
		ADOBind_comEDIT.UpdateControls()
	End Sub
	Public Sub VB6_RemoveADODataBinding()
		ADOBind_comEDIT.Clear()
		ADOBind_comEDIT.Dispose()
		ADOBind_comEDIT = Nothing
	End Sub
End Class
```

Uppgraderingsrapporten från Visual Basic .net.

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
(Global Issues)				0	0	0
Global update issues:						
None						
deEDIT.vb	deEDIT.Dsr	ActiveX Designer	Upgraded	0	0	0
Upgrade Issues for deEDIT.Dsr:						

None

formEDIT.vb	formEDIT.frm	Form	Upgraded with issues	9	2	11

Upgrade Issues for formEDIT.frm:

#	Severity	Location	Object Type	Object Name	Property	Description
1	Design Error	(Layout)	TextBox	Text3	DataField	TextBox property Text3.DataField was not upgraded.
2	Design Error	(Layout)	TextBox	Text3	DataMember	TextBox property Text3.DataMember was not upgraded.
3	Design Error	(Layout)	TextBox	Text3	DataSource	TextBox property Text3.DataSource was not upgraded.
4	Design Error	(Layout)	TextBox	txtEMAIL	DataField	TextBox property txtEMAIL.DataField was not upgraded.
5	Design Error	(Layout)	TextBox	txtEMAIL	DataMember	TextBox property txtEMAIL.DataMember was not upgraded.
6	Design Error	(Layout)	TextBox	txtEMAIL	DataSource	TextBox property txtEMAIL.DataSource was not upgraded.
7	Design Error	(Layout)	TextBox	txtNAMN	DataField	TextBox property txtNAMN.DataField was not upgraded.
8	Design Error	(Layout)	TextBox	txtNAMN	DataMember	TextBox property txtNAMN.DataMember was not upgraded.
9	Design Error	(Layout)	TextBox	txtNAMN	DataSource	TextBox property txtNAMN.DataSource was not upgraded.
10	Runtime Warning	butLÄGGTILL_Click				Couldn't resolve default property of object deEDIT.comINSERT.
11	Runtime Warning	butTABORT_Click				Couldn't resolve default property of

						object deEDIT.comDELETE.
2 File(s)		ActiveX Designers: 1 Forms: 1	Upgraded: 2 Not upgraded: 0	9	2	11

Click here for help with troubleshooting upgraded projects

Upgrade Settings

LogFile: proEDIT.log

MigrateProjectTo: WinExe

OutputDir: D:\Skolan\Xjobb\VB.net program\EDIT

OutputName: proEDIT.vbproj

ProjectName: proEDIT

ProjectPath: D:\Skolan\Xjobb\VB6program\EDIT\proEDIT.vbp

Genomgång av ramverket

Fas 0: Förbered migrationen

0.1 Fastställ krav för mätning av hur bra migrationen gått.

Migrationsprocessen ska, med hjälp av den wizard som finns tillgänglig i Visual Studio.net, problemfritt kunna migrera gränssnittet samt applikationen till det nya Visual Basic .net. Detta ska ske utan att några ytterligare modifikationer ska behöva göras för att upprätthålla gränssnittet eller funktionaliteten i applikationen.

0.2 Fastställ hur det nya systemet ska byggas upp

En koppling till en databas på en SQLserver ska ske via en OLDBprovider. Det ska förutom de funktioner som applikationen Bläddra hade kunna gå att lägga till samt ta bort en förutbestämd post i tabellen.

Fas 1: Förstå semantiken i det äldre systemet

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.

Gränssnittet är likvärdigt det i Bläddra fast textsträngarna har tagits bort och de två knapparna Ta bort och Lägg till har lagts till i formuläret. Placeringen på de olika objekten har ändrats något i jämförelse med Bläddra. Under de tre textrutorna så befinner sig knapparna Lägg till samt Ta bort och till höger om dessa Stäng knappen. Längst upp till höger så befinner sig nu de två knapparna Förra och Nästa.

Några överflödigheter i gränssnittet kan inte identifieras.

Gränssnittet ska vara likvärdigt det i den äldre applikationen vad gäller färg, storlek, typsnitt samt placering dels på objekten i formuläret och dels på formuläret i sig.

1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

Applikationen funktionalitet är den samma som i Bläddra med ändringen att det tillkommit två knappar som kan lägga till alternativt ta bort en förutbestämd post i tabellen som applikationen kopplas till. Detta sker genom att de genom programmeringsspråket kopplas till de två kommandona comINSERT respektive comDELETE.

Några överflödigheter kan inte identifieras.

Den nya applikationen ska på ett likvärdigt sätt kunna utföra de funktionaliteter som den äldre applikationen gjorde.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ funktionerna för de nya data.

I applikationen så används tabellen Leverantör som har nyckeln namn samt de två kolumnerna E-mail samt telefon.

Inga överflödigheter kunde identifieras.

Funktionerna för den nya datan ska vara likvärdig den i det äldre.

1.5 Slutför migrationskraven

Inga ytterligare migrationskrav kunde identifieras.

Fas 2: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen

2.1 Migrera det äldre gränssnittet och applikationen

Migrationen genomfördes med hjälp av den wizard som finns inbyggd i Visual Studio .net.

2.1.1 Testa gränssnittet mot data

Tester av gränssnittet utfördes.

2.1.2 Bekräfta gentemot kraven på gränssnittet

Gränssnittets typsnitt ändrades men ingen information försvann**.**

2.1.3 Rätta till fel som uppstått i gränssnittet

Felen rättades till genom att utöka rubrikens ram.

2.1.4 Testa applikationen mot data

Tester av applikationen utfördes.

2.1.5 Bekräfta gentemot kraven på applikationen

Applikationen fungerade korrekt. All data som skulle visar och ändras gjorde detta på ett tillförlitlig sätt.

2.1.6 Rätta till fel som uppstått i applikationen

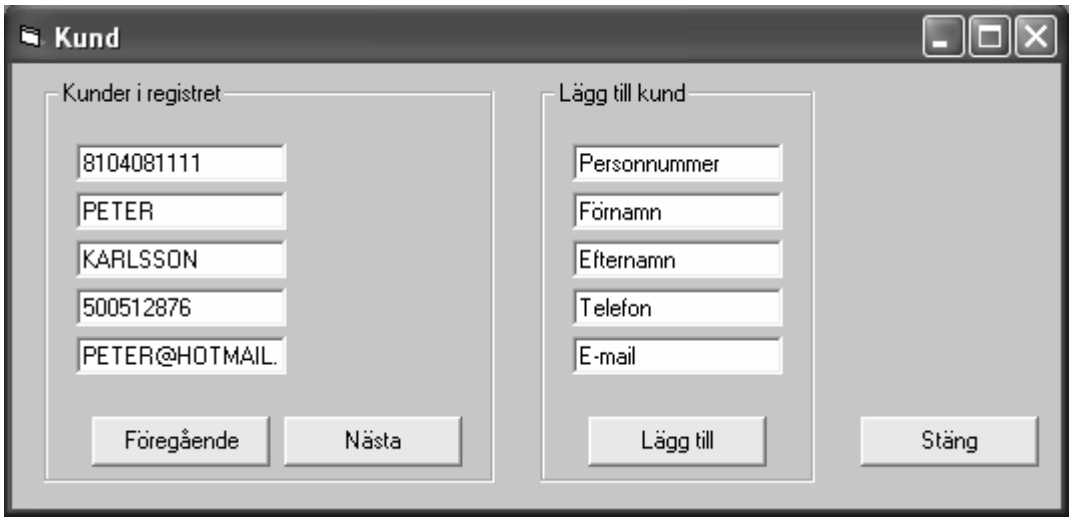
Inga fel hade uppstått.

2.2 Träna användarna på de nya applikationerna

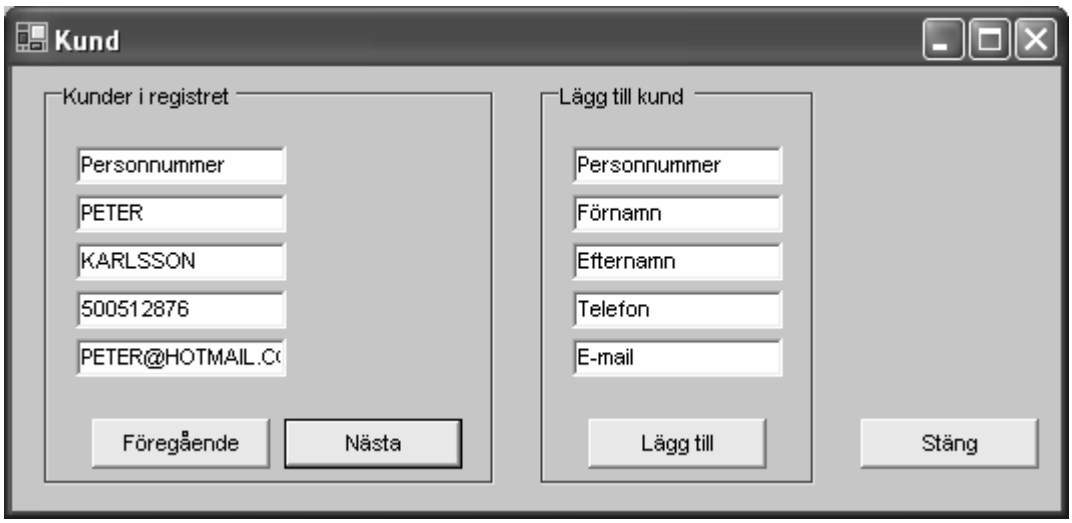
Inga användare finns till denna applikation varpå ingen träning genomfördes.

Appendix 4: Kund

Gränssnittet.



(Bild 1: Kund i Visual Basic 6)



(Bild 2: Kund i Visual Basic .net)

Koden.

Visual Basic 6:

```
Private Sub butFÖREGÅENDE_Click()
deKUND.rsComKUND.MovePrevious
End Sub

Private Sub butLÄGGTILL_Click()
deKUND.rsComKUND.AddNew
```

```
deKUND.rsComKUND.Fields("PERSONNUMMER") = txtPNR.Text
deKUND.rsComKUND.Fields("FÖRNAMN") = txtFNAMN.Text
deKUND.rsComKUND.Fields("EFTERNAMN") = txtENAMN.Text
deKUND.rsComKUND.Fields("KUNDTELEFON") = txtTELE.Text
deKUND.rsComKUND.Fields("KUNDEMAIL") = txtMAIL.Text
deKUND.rsComKUND.Update
txtPNR.Text = "Personnummer"
txtFNAMN.Text = "Förnamn"
txtENAMN.Text = "Efternamn"
txtTELE.Text = "Telefon"
txtMAIL.Text = "E-mail"
End Sub

Private Sub butNÄSTA_Click()
deKUND.rsComKUND.MoveNext
End Sub

Private Sub butSTÄNG_Click()
End
End Sub
```

Visual Basic .net

```
Option Strict Off
Option Explicit On
    Private Sub butFÖREGÅENDE_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles butFÖREGÅENDE.Click

        deKUND.rsComKUND.MovePrevious()

    End Sub

    Private Sub butLÄGGTILL_Click(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles butLÄGGTILL.Click

        deKUND.rsComKUND.AddNew()
        deKUND.rsComKUND.Fields("PERSONNUMMER").Value = txtPNR.Text
```

```
            deKUND.rsComKUND.Fields("FÖRNAMN").Value = txtFNAMN.Text
            deKUND.rsComKUND.Fields("EFTERNAMN").Value              =
txtENAMN.Text
            deKUND.rsComKUND.Fields("KUNDTELEFON").Value            =
txtTELE.Text
            deKUND.rsComKUND.Fields("KUNDEMAIL").Value = txtMAIL.Text
            deKUND.rsComKUND.Update()
            txtPNR.Text = "Personnummer"
            txtFNAMN.Text = "Förnamn"
            txtENAMN.Text = "Efternamn"
            txtTELE.Text = "Telefon"
            txtMAIL.Text = "E-mail"

      End Sub

      Private Sub butNÄSTA_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles butNÄSTA.Click

            deKUND.rsComKUND.MoveNext()

      End Sub

      Private Sub butSTÄNG_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles butSTÄNG.Click

            End

      End Sub
      Public Sub VB6_AddADODataBinding()
            deKUND.ComKUND()
            ADOBind_ComKUND = New VB6.MBindingCollection()
            ADOBind_ComKUND.DataSource = deKUND
            ADOBind_ComKUND.DataMember = "ComKUND"
            ADOBind_ComKUND.Add(txtEMAIL,     "Text",     "KUNDEMAIL",
Nothing, "txtEMAIL")
            ADOBind_ComKUND.Add(txtTELEFON,   "Text",   "KUNDTELEFON",
Nothing, "txtTELEFON")
            ADOBind_ComKUND.Add(txtEFTERNAMN,   "Text",   "EFTERNAMN",
Nothing, "txtEFTERNAMN")
            ADOBind_ComKUND.Add(txtFÖRNAMN,     "Text",     "FÖRNAMN",
Nothing, "txtFÖRNAMN")
            ADOBind_ComKUND.UpdateMode                              =
VB6.UpdateMode.vbUpdateWhenPropertyChanges
            ADOBind_ComKUND.UpdateControls()
```

```
        End Sub
        Public Sub VB6_RemoveADODataBinding()
            ADOBind_ComKUND.Clear()
            ADOBind_ComKUND.Dispose()
            ADOBind_ComKUND = Nothing
        End Sub
    End Class
```

Uppgraderingsrapporten från Visual Basic .net.

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
⊞ (Global Issues)				0	0	0
Global update issues:						
None						
⊞ deKUND.vb	deKUND.Dsr	ActiveX Designer	Upgraded	0	0	0
Upgrade Issues for deKUND.Dsr:						
None						
⊟ frmKUND.vb	frmKUND.frm	Form	Upgraded with issues	15	0	15
Upgrade Issues for frmKUND.frm:						

#	Severity	Location	Object Type	Object Name	Property	Description
1	Design Error	(Layout)	TextBox	txtEFTERNAMN	DataField	TextBox proper txtEFTERNAMN.DataField w not upgraded.
2	Design Error	(Layout)	TextBox	txtEFTERNAMN	DataMember	TextBox proper txtEFTERNAMN.DataMember w not upgraded.
3	Design Error	(Layout)	TextBox	txtEFTERNAMN	DataSource	TextBox proper txtEFTERNAMN.DataSource w not upgraded.
4	Design Error	(Layout)	TextBox	txtEMAIL	DataField	TextBox proper txtEMAIL.DataField was r upgraded.
5	Design Error	(Layout)	TextBox	txtEMAIL	DataMember	TextBox proper

						txtEMAIL.DataMember was r upgraded.
6	Design Error	(Layout)	TextBox	txtEMAIL	DataSource	TextBox proper txtEMAIL.DataSource was r upgraded.
7	Design Error	(Layout)	TextBox	txtFÖRNAMN	DataField	TextBox proper txtFÖRNAMN.DataField was r upgraded.
8	Design Error	(Layout)	TextBox	txtFÖRNAMN	DataMember	TextBox proper txtFÖRNAMN.DataMember w not upgraded.
9	Design Error	(Layout)	TextBox	txtFÖRNAMN	DataSource	TextBox proper txtFÖRNAMN.DataSource w not upgraded.
10	Design Error	(Layout)	TextBox	txtPERSONNUMMER	DataField	TextBox proper txtPERSONNUMMER.DataField was not upgraded.
11	Design Error	(Layout)	TextBox	txtPERSONNUMMER	DataMember	TextBox proper txtPERSONNUMMER.DataMemb was not upgraded.
12	Design Error	(Layout)	TextBox	txtPERSONNUMMER	DataSource	TextBox proper txtPERSONNUMMER.DataSourc was not upgraded.
13	Design Error	(Layout)	TextBox	txtTELEFON	DataField	TextBox proper txtTELEFON.DataField was r upgraded.
14	Design Error	(Layout)	TextBox	txtTELEFON	DataMember	TextBox proper txtTELEFON.DataMember w not upgraded.
15	Design Error	(Layout)	TextBox	txtTELEFON	DataSource	TextBox proper txtTELEFON.DataSource was r upgraded.

2 File(s)	ActiveX Designers: 1 Forms: 1	Upgraded: 2 Not upgraded: 0	15	0	15

Click here for help with troubleshooting upgraded projects

Upgrade Settings

LogFile: Kund.log

MigrateProjectTo: WinExe

OutputDir: D:\Skolan\Xjobb\VB.net program\KUND2

OutputName: Kund.vbproj

ProjectName: Kund

ProjectPath: D:\Skolan\Xjobb\VB6program\Kund\Kund.vbp

Genomgång av ramverket.

Fas 0: Förbered migrationen

0.1 Fastställ krav för mätning av hur bra migrationen gått.

Migrationsprocessen ska, med hjälp av den wizard som finns tillgänglig i Visual Studio.net, problemfritt kunna migrera gränssnittet samt applikationen till det nya Visual Basic .net. Detta ska ske utan att några ytterligare modifikationer ska behöva göras för att upprätthålla gränssnittet eller funktionaliteten i applikationen.

0.2 Fastställ hur det nya systemet ska byggas upp

Systemet ska med hjälp av en OLDBprovider koppla applikationen till en databas via en SQLserver. Applikationen ska med hjälp av programering kunna lägga till i en tabell.

Fas 1: Förstå semantiken i det äldre systemet

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.

Gränssnittet är uppbyggt på två ramar vilken den första Kunder i registret innehåller textrutor innehållande den information som kommer ut ur applikationen gällande data som finns i tabellen Kund (se Appendix B3). Denna ram innehåller även de två knapparna Förra och Nästa. Den andra ramen innehåller textrutor som används för att användaren ska kunna föra in data till systemet samt en knapp för att kunna utföra detta. Det finns även en ytterligare knapp utanför dessa två ramarna som stänger applikationen.

Inga överflödigheter identifierades.

Gränssnittet ska vara likvärdigt med det i den äldre applikationen vad gäller färg, form, placering och finter på formuläret samt dess komponenter.

1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

Applikationen har dels samma funktion som Bläddra där användaren genom två knappar kan navigera fram och tillbaka mellan en tabells poster. I denna applikation så går det även att skriva in till systemet vad som ska lagras i tabellen, detta görs genom textrutor och knappen Lägg till. Kopplingen mellan textrutorna där användaren skriver in vad som ska lagras och vart i databasen detta ska lagras görs med hjälp av Visual Basic 6 kod.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ funktionerna för de nya data.

Data hämtas från tabellen Kund tabellen i databasen Northwind. Tabellen består av nyckeln personnummer samt förnamn, efternamn, telefon och E-mail. Det är även i denna tabellen som data lagras.

Inga överflödigheter kunde identifieras.

Funktionerna för den nya datan ska vara likvärdig den i det äldre.

1.5 Slutför migrationskraven

Inga ytterligare migrationskrav kunde identifieras.

Fas 2: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen

2.1 Migrera det äldre gränssnittet och applikationen

Migrationen genomfördes med hjälp av den wizard som finns inbyggd i Visual Studio .net.

2.1.1 Testa gränssnittet mot data

Tester av gränssnittet utfördes.

2.1.2 Bekräfta gentemot kraven på gränssnittet

Gränssnittet hade samma fel som applikationen Bläddra.

2.1.3 Rätta till fel som uppstått i gränssnittet

Felen rättades till genom att utöka rubrikens ram.

2.1.4 Testa applikationen mot data

Tester av applikationen utfördes.

2.1.5 Bekräfta gentemot kraven på applikationen

Personnumret på kunderna i tabellen kommer inte med i den textruta som är tänkt till detta, vilket det gjorde i den äldre versionen. Inte heller tabordningen stämde överens med den äldre versionens. Detta ledde till att kraven inte uppfylldes.

2.1.6 Rätta till fel som uppstått i applikationen

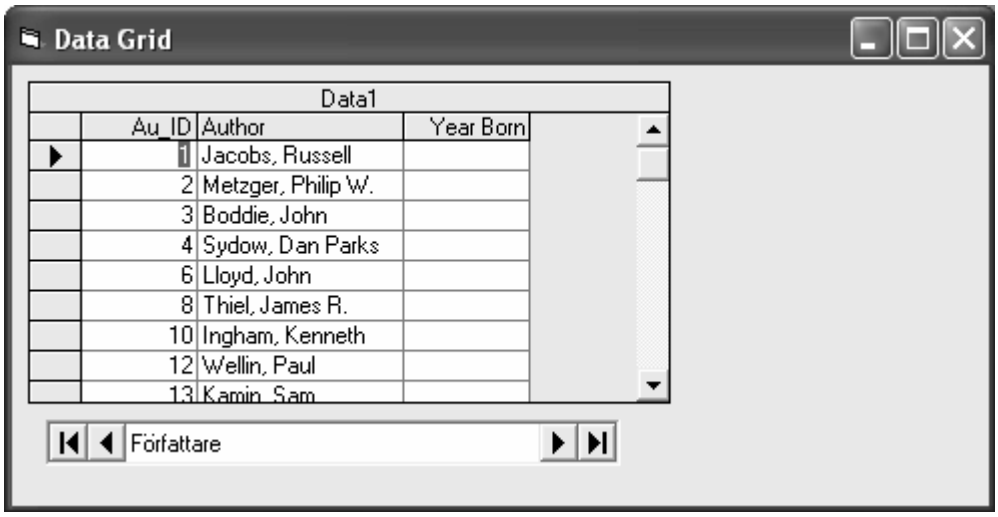
Uppgraderingsrapporten gav inga indikationer på vad felen kunde vara varpå felet ansågs så invecklat att inga ytterligare försök till att rätta till det gjordes.

2.2 Träna användarna på de nya applikationerna

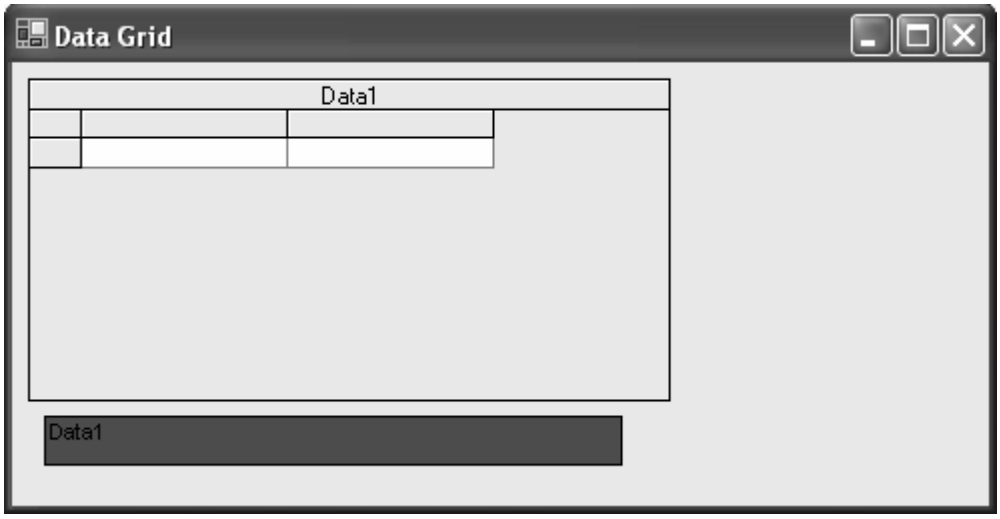
Inga användare finns till denna applikation varpå ingen träning genomfördes.

Appendix 5: Data Grid

Gränssnitet.



(Bild 1: Data Grid i Visual Basic 6)



(Bild 2:Data Grid i Visual Basic .net)

Koden.

Visual Basic 6:

Applikationen i Visual Basic 6 gjordes utan kod.

Visual Basic .net:

```
Ingen kod migrerades
```

Uppgraderingsrapporten från Visual Basic .net.

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
(Global Issues)				0	0	0
Global update issues:						
None						
formDATAGRID.vb	formDATAGRID.frm	Form	Upgraded with issues	3	0	3
Upgrade Issues for formDATAGRID.frm:						
#	Severity	Location	Object Type	Object Name	Property	Description
1	Design Error	(Layout)				Data object was not upgraded.
2	Design Error	(Layout)	Data	Data1		Data control Data1 was not upgraded.
3	Design Error	(Layout)	DBGrid	DBGrid1	Bindings	DBGrid property DBGrid1.Bindings was not upgraded.
1 File(s)			Forms: 1	Upgraded: 1 Not upgraded: 0	3	0	3

Click here for help with troubleshooting upgraded projects

Upgrade Settings

LogFile: proDATAGRID.log

MigrateProjectTo: WinExe

OutputDir: D:\Skolan\Xjobb\VB.net program\GRID

OutputName: proDATAGRID.vbproj

ProjectName: proDATAGRID

ProjectPath: D:\Skolan\Xjobb\VB6program\GRID\proDATAGRID.vbp

Genomgång av ramverket.

Fas 0: Förbered migrationen

0.1 Fastställ krav för mätning av hur bra migrationen gått.

Migrationsprocessen ska, med hjälp av den wizard som finns tillgänglig i Visual Studio.net, problemfritt kunna migrera gränssnittet samt applikationen till det nya Visual Basic .net. Detta ska ske utan att några ytterligare modifikationer ska behöva göras för att upprätthålla gränssnittet eller funktionaliteten i applikationen.

0.2 Fastställ hur det nya systemet ska byggas upp

Systemet ska byggas på en koppling mellan en MDB databas och en Datagrid via en Datakontroll. Kopplingen sker via Datagriddens och Datakontrollens properties. Data från tabellen Authors ska visas i Datagridden och via Datakontrollen ska det gå att navigera ibland posterna.

Fas 1: Förstå semantiken i det äldre systemet

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.

Gränssnittet består endast av Datakontrollen som är belägen under Datagridden som är formulärets andra objekt.

Några överflödigheter kunde inte identifieras.

Det nya gränssnittet ska vara likvärdigt med det äldre när det gäller färg, form, placering samt typsnitt.

1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

Användaren ska kunna navigera genom tabellen Authors med hjälp av Datakontrollen. Med den ska det gå att förflytta sig fram och tillbaka ett steg samt kunna gå att hoppa sist respektive först bland posterna i tabellen. Data ska visas i Datagridden.

Några överflödigheter kunde inte identifieras.

Den nya applikationen ska på ett likvärdigt sätt kunna utföra de funktioner som den äldre applikationen kunde göra.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ funktionerna för de nya data.

Data till denna applikation hämtades ur en MDBdatabas som heter Biblio och som ofta följer med Visual Studio programmen. I databasen så används endast tabellen Authors som består av författarnas namn samt deras IDnummer.

Några överflödigheter kunde inte identifieras.

Den nya data som används ska visas på samma sätt och innehålla samma information. Ingen data ska heller gå förlorad under migrationen.

1.5 Slutför migrationskraven

Inga ytterligare migrationskrav kunde identifieras.

Fas 2: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen

2.1 Migrera det äldre gränssnittet och applikationen

Migrationen genomfördes med hjälp av den wizard som finns inbyggd i Visual

Studio .net.

2.1.1 Testa gränssnittet mot data

2.1.2 Bekräfta gentemot kraven på gränssnittet

Applikationen kunde inte köras på det sätt som önskades vilket medförde att grässnittet inte går att utvärdera.

2.1.3 Rätta till fel som uppstått i gränssnittet

Inget gjordes för att rätta till de fel som uppstått.

2.1.4 Testa applikationen mot data

2.1.5 Bekräfta gentemot kraven på applikationen

Applikationen kunde köras men funktionen att koppla dat via Datakontrollen fungerade inte. Ingen data visades därmed i Datagridden.

2.1.6 Rätta till fel som uppstått i applikationen

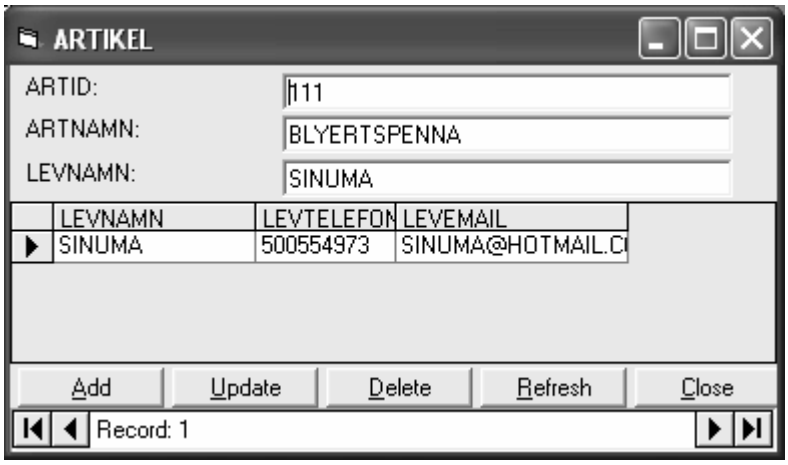
Felen ansågs så stora att några försök att rätta till dessa inte gjordes.

2.2 Träna användarna på de nya applikationerna

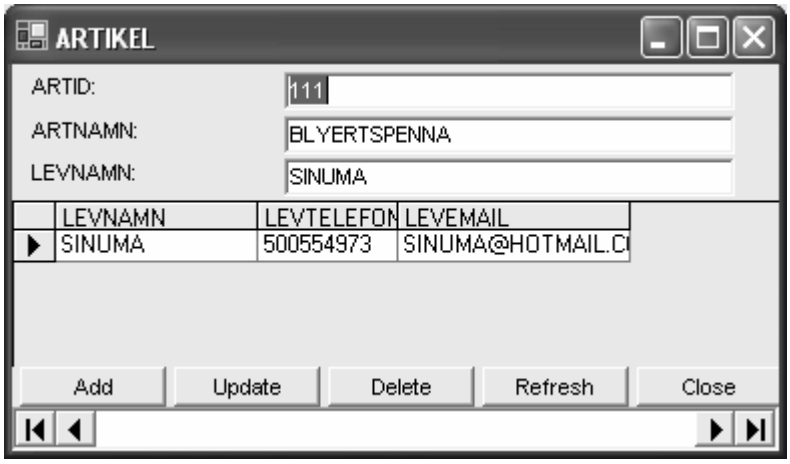
Inga användare finns till denna applikation varpå ingen träning genomfördes.

Appendix 6: Artikel

Gränssnittet.



(Bild 1: Artikel i Visual Basic 6)



(Bild 2: Artikel i Visual Basic .net)

Koden.

Visual Basic 6:

```
Private Sub Form_Load()
  Set grdDataGrid.DataSource =
datPrimaryRS.Recordset("ChildCMD").UnderlyingValue
End Sub

Private Sub Form_Resize()
  On Error Resume Next
  'This will resize the grid when the form is resized
  grdDataGrid.Width = Me.ScaleWidth
```

```
  grdDataGrid.Height = Me.ScaleHeight - grdDataGrid.Top - datPrimaryRS.Height - 30 - picButtons.Height
End Sub

Private Sub Form_Unload(Cancel As Integer)
  Screen.MousePointer = vbDefault
End Sub

Private Sub datPrimaryRS_Error(ByVal ErrorNumber As Long, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, fCancelDisplay As Boolean)
  'This is where you would put error handling code
  'If you want to ignore errors, comment out the next line
  'If you want to trap them, add code here to handle them
  MsgBox "Data error event hit err:" & Description
End Sub

Private Sub datPrimaryRS_MoveComplete(ByVal adReason As ADODB.EventReasonEnum, ByVal pError As ADODB.Error, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
  'This will display the current record position for this recordset
  datPrimaryRS.Caption = "Record: " & CStr(datPrimaryRS.Recordset.AbsolutePosition)
End Sub

Private Sub datPrimaryRS_WillChangeRecord(ByVal adReason As ADODB.EventReasonEnum, ByVal cRecords As Long, adStatus As ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset)
  'This is where you put validation code
  'This event gets called when the following actions occur
  Dim bCancel As Boolean
  Select Case adReason
  Case adRsnAddNew
  Case adRsnClose
  Case adRsnDelete
  Case adRsnFirstChange
  Case adRsnMove
  Case adRsnRequery
```

```
  Case adRsnResynch
  Case adRsnUndoAddNew
  Case adRsnUndoDelete
  Case adRsnUndoUpdate
  Case adRsnUpdate
  End Select
  If bCancel Then adStatus = adStatusCancel
End Sub

Private Sub cmdAdd_Click()
  On Error GoTo AddErr
  datPrimaryRS.Recordset.AddNew
  Exit Sub
AddErr:
  MsgBox Err.Description
End Sub

Private Sub cmdDelete_Click()
  On Error GoTo DeleteErr
  With datPrimaryRS.Recordset
    .Delete
    .MoveNext
    If .EOF Then .MoveLast
  End With
  Exit Sub
DeleteErr:
  MsgBox Err.Description
End Sub

Private Sub cmdRefresh_Click()
  'This is only needed for multi user apps
  On Error GoTo RefreshErr
  datPrimaryRS.Refresh
  Set grdDataGrid.DataSource =
datPrimaryRS.Recordset("ChildCMD").UnderlyingValue
  Exit Sub
```

```
RefreshErr:
  MsgBox Err.Description
End Sub

Private Sub cmdUpdate_Click()
  On Error GoTo UpdateErr
  datPrimaryRS.Recordset.UpdateBatch adAffectAll
  Exit Sub
UpdateErr:
  MsgBox Err.Description
End Sub

Private Sub cmdClose_Click()
  Unload Me
End Sub
```

Visual Basic .net:

```
Option Strict Off
Option Explicit On
Friend Class formKOPPLING
	Inherits System.Windows.Forms.Form
	Private Sub formKOPPLING_Load(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles MyBase.Load
		grdDataGrid.DataSource = datPrimaryRS.Recordset.Fields("ChildCMD").UnderlyingValue
	End Sub


	'UPGRADE_WARNING: Event formKOPPLING.Resize may fire when form is initialized. Click for more: 'ms-help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup2075"'
	Private Sub formKOPPLING_Resize(ByVal eventSender As System.Object, ByVal eventArgs As System.EventArgs) Handles MyBase.Resize
		On Error Resume Next
		'This will resize the grid when the form is resized
		grdDataGrid.Width = Me.ClientRectangle.Width
		grdDataGrid.Height = VB6.TwipsToPixelsY(VB6.PixelsToTwipsY(Me.ClientRectangle.Height) - VB6.PixelsToTwipsY(grdDataGrid.Top) - VB6.PixelsToTwipsY(datPrimaryRS.Height) - 30 - VB6.PixelsToTwipsY(picButtons.Height))
```

```
    End Sub

    'UPGRADE_WARNING: Form event formKOPPLING.Unload has a new
behavior. Click for more: 'ms-
help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup2065"'
    Private Sub formKOPPLING_Closed(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
MyBase.Closed
        'UPGRADE_WARNING: Screen property Screen.MousePointer has
a new behavior. Click for more: 'ms-
help://MS.VSCC.2003/commoner/redir/redirect.htm?keyword="vbup2065"'
        System.Windows.Forms.Cursor.Current =
System.Windows.Forms.Cursors.Default
    End Sub

    Private Sub datPrimaryRS_Error(ByVal ErrorNumber As Integer,
ByRef Description As String, ByVal Scode As Integer, ByVal Source As
String, ByVal HelpFile As String, ByVal HelpContext As Integer, ByRef
fCancelDisplay As Boolean) Handles datPrimaryRS.Error
        'This is where you would put error handling code
        'If you want to ignore errors, comment out the next line
        'If you want to trap them, add code here to handle them
        MsgBox("Data error event hit err:" & Description)
    End Sub

    Private Sub datPrimaryRS_MoveComplete(ByVal adReason As
ADODB.EventReasonEnum, ByVal pError As ADODB.Error, ByRef adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset) Handles
datPrimaryRS.MoveComplete
        'This will display the current record position for this
recordset
        datPrimaryRS.Text = "Record: " &
CStr(datPrimaryRS.Recordset.AbsolutePosition)
    End Sub

    Private Sub datPrimaryRS_WillChangeRecord(ByVal adReason As
ADODB.EventReasonEnum, ByVal cRecords As Integer, ByRef adStatus As
ADODB.EventStatusEnum, ByVal pRecordset As ADODB.Recordset) Handles
datPrimaryRS.WillChangeRecord
        'This is where you put validation code
        'This event gets called when the following actions occur
        Dim bCancel As Boolean

        Select Case adReason
            Case ADODB.EventReasonEnum.adRsnAddNew
            Case ADODB.EventReasonEnum.adRsnClose
            Case ADODB.EventReasonEnum.adRsnDelete
```

```
                  Case ADODB.EventReasonEnum.adRsnFirstChange
                  Case ADODB.EventReasonEnum.adRsnMove
                  Case ADODB.EventReasonEnum.adRsnRequery
                  Case ADODB.EventReasonEnum.adRsnResynch
                  Case ADODB.EventReasonEnum.adRsnUndoAddNew
                  Case ADODB.EventReasonEnum.adRsnUndoDelete
                  Case ADODB.EventReasonEnum.adRsnUndoUpdate
                  Case ADODB.EventReasonEnum.adRsnUpdate
            End Select

            If bCancel Then adStatus =
ADODB.EventStatusEnum.adStatusCancel
      End Sub

      Private Sub cmdAdd_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles cmdAdd.Click
            On Error GoTo AddErr
            datPrimaryRS.Recordset.AddNew()

            Exit Sub
AddErr:
            MsgBox(Err.Description)
      End Sub

      Private Sub cmdDelete_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles cmdDelete.Click
            On Error GoTo DeleteErr
            With datPrimaryRS.Recordset
                  .Delete()
                  .MoveNext()
                  If .EOF Then .MoveLast()
            End With
            Exit Sub
DeleteErr:
            MsgBox(Err.Description)
      End Sub

      Private Sub cmdRefresh_Click(ByVal eventSender As
System.Object, ByVal eventArgs As System.EventArgs) Handles
cmdRefresh.Click
            'This is only needed for multi user apps
            On Error GoTo RefreshErr
```

```
            datPrimaryRS.Refresh()
            grdDataGrid.DataSource                                    =
datPrimaryRS.Recordset.Fields("ChildCMD").UnderlyingValue
            Exit Sub
RefreshErr:
            MsgBox(Err.Description)
      End Sub


      Private Sub cmdUpdate_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles cmdUpdate.Click
            On Error GoTo UpdateErr



      datPrimaryRS.Recordset.UpdateBatch(ADODB.AffectEnum.adAffectAll
)
            Exit Sub
UpdateErr:
            MsgBox(Err.Description)
      End Sub


      Private Sub cmdClose_Click(ByVal eventSender As System.Object,
ByVal eventArgs As System.EventArgs) Handles cmdClose.Click
            Me.Close()
      End Sub
      Public Sub VB6_AddADODataBinding()
            ADOBind_datPrimaryRS = New VB6.MBindingCollection()
            ADOBind_datPrimaryRS.DataSource   =   CType(datPrimaryRS,
msdatasrc.DataSource)
            ADOBind_datPrimaryRS.Add(_txtFields_2, "Text", "LEVNAMN",
Nothing, "_txtFields_2")
            ADOBind_datPrimaryRS.Add(_txtFields_1, "Text", "ARTNAMN",
Nothing, "_txtFields_1")
            ADOBind_datPrimaryRS.Add(_txtFields_0,  "Text",  "ARTID",
Nothing, "_txtFields_0")
            ADOBind_datPrimaryRS.UpdateMode                           =
VB6.UpdateMode.vbUpdateWhenPropertyChanges
            ADOBind_datPrimaryRS.UpdateControls()
      End Sub
      Public Sub VB6_RemoveADODataBinding()
            ADOBind_datPrimaryRS.Clear()
            ADOBind_datPrimaryRS.Dispose()
            ADOBind_datPrimaryRS = Nothing
      End Sub
End Class
```

Uppgraderingsrapporten från Visual Studio .net.

New Filename	Original Filename	File Type	Status	Errors	Warnings	Total Issues
(Global Issues)				0	0	0

Global update issues:

None

formKOPPLING.vb	formKOPPLING.frm	Form	Upgraded with issues	8	3	11

Upgrade Issues for formKOPPLING.frm:

#	Severity	Location	Object Type	Object Name	Property	Description
1	Design Error	(Layout)	Adodc	datPrimaryRS	Connect	Adodc property datPrimaryRS.Connect was not upgraded.
2	Design Error	(Layout)	TextBox	_txtFields_0	DataField	TextBox property _txtFields_0.DataField was not upgraded.
3	Design Error	(Layout)	TextBox	_txtFields_0	DataSource	TextBox property _txtFields_0.DataSource was not upgraded.
4	Design Error	(Layout)	TextBox	_txtFields_1	DataField	TextBox property _txtFields_1.DataField was not upgraded.
5	Design Error	(Layout)	TextBox	_txtFields_1	DataSource	TextBox property _txtFields_1.DataSource was not upgraded.
6	Design Error	(Layout)	TextBox	_txtFields_2	DataField	TextBox property _txtFields_2.DataField was not upgraded.
7	Design Error	(Layout)	TextBox	_txtFields_2	DataSource	TextBox property _txtFields_2.DataSource was not upgraded.
8	Design Error	(Layout)	VBControlExtender	datPrimaryRS	Password	VBControlExtender method datPrimaryRS.Password was not upgraded.
9	Runtime Warning	Form_Resize	Form		Resize	Event formKOPPLING.Resize

						may fire when form is initialized.
10	Runtime Warning	Form_Unload	Form	formKOPPLING	Unload	Form event formKOPPLING.Unload has a new behavior.
11	Runtime Warning	Form_Unload	Screen	Screen	MousePointer	Screen property Screen.MousePointer has a new behavior.

1 File(s)		Forms: 1	Upgraded: 1 Not upgraded: 0	8	3	11

Click here for help with troubleshooting upgraded projects

Upgrade Settings

LogFile: proKOPPLING.log

MigrateProjectTo: WinExe

OutputDir: D:\Skolan\Xjobb\VB.net program\KOPPLING

OutputName: proKOPPLING.vbproj

ProjectName: proKOPPLING

ProjectPath: D:\Skolan\Xjobb\VB6program\KOPPLING\proKOPPLING.vbp

Genomgång av ramverket.

Fas 0: Förbered migrationen

0.1 Fastställ krav för mätning av hur bra migrationen gått.

Migrationsprocessen ska, med hjälp av den wizard som finns tillgänglig i Visual Studio.net, problemfritt kunna migrera gränssnittet samt applikationen till det nya Visual Basic .net. Detta ska ske utan att några ytterligare modifikationer ska behöva göras för att upprätthålla gränssnittet eller funktionaliteten i applikationen.

0.2 Fastställ hur det nya systemet ska byggas upp

Systemet ska bygas upp så att användaren via en Datakontroll kan bläddra i tabellen Artikel som finns i databasen Northwind på en lokal SQLserver. Data från tabellen visas i tre stycken textrutor. Tabellen Artikel har ett en-till-många förhållande till Leverantör. Leverantören som är kopplad till den aktuella posten i tabellen visas i en Datagrid. Det går även med hjälp av knappar att lägga till (Add), uppdatera (Update), ta bort (Delete), uppdatera databasen (Refresh) samt stänga applikationen (Close).

Fas 1: Förstå semantiken i det äldre systemet

1.1 Skaffa förståelse för det äldre gränssnittet, identifiera överflödigheter och fastställ funktionerna för det nya gränssnittet.

Gränssnittet består av tre textrutor med tillhörande text där data från tabellen Artikel

visas. Under dessa finns en Datagrid där data från tabellen Leverantör visas. Under dessa kommer en rad med de kommandoknappar sim finns i appliktionen. Längst ner i formuläret finns Datakontrollen.

Några överflödigheter kunde inte identifieras.

Det nya gränssnittet ska vara likvärdigt med det äldre när det gäller färg, form, placering samt typsnitt.

1.2 Skaffa förståelse för den äldre applikationen, identifiera överflödigheter och fastställ funktionerna för den nya applikationen.

Applikationen ska visa data från dels tabellen Artikel och dels från Leverantör. Genom datakontrollen ska användaren kunna bläddra bland artiklarna medans Leverantörs data i datagridden uppdateras automatiskt. Användaren ska kunna lägga till, ta bort samt ändra i tabellen Atikel. Det ska även gå att uppdatera databasen, samt stänga applikationen.

Några överflödigheter kunde inte identifieras.

Den nya applikationen ska på ett likvärdigt sätt kunna utföra de funktioner som den äldre applikationen kunde göra.

1.3 Skaffa förståelse för de äldre data, identifiera överflödigheter och fastställ funktionerna för de nya data.

Data till denna appliaktion kommer från två tabellerna Artikel och Leverantör, belägna i databasen Northwind på en SQLserver. Det finnns ett en-till-många förhållande mellan dessa två som gör att varje artikel har en speiell leverantör och det är den som ska visas i datagriden.

Några överflödigheter kunde inte identifieras.

Den nya data som används ska visas på samma sätt och innehålla samma information. Ingen data ska heller gå förlorad under migrationen.

1.5 Slutför migrationskraven

Inga ytterligare migrationskrav kunde identifieras.

Fas 2: Migrera alla komponenter (inte data) i det äldre systemet till den nya arkitekturen

2.1 Migrera det äldre gränssnittet och applikationen

Migrationen genomfördes med hjälp av den wizard som finns inbyggd i Visual Studio .net.

2.1.1 Testa gränssnittet mot data

2.1.2 Bekräfta gentemot kraven på gränssnittet

Applikationens gränssnitt stämde överens med de krav som ställts på den.

2.1.3 Rätta till fel som uppstått i gränssnittet

Inga fel hade uppstått.

2.1.4 Testa applikationen mot data

2.1.5 Bekräfta gentemot kraven på applikationen

Applikationen fungerade som den skulle utifrån de krav som ställts på den.

2.1.6 Rätta till fel som uppstått i applikationen

Inga fel hade uppstått.

2.2 Träna användarna på de nya applikationerna

Inga användare finns till denna applikation varpå ingen träning genomfördes.