

## **Uppdatering av program för utskrift av rutintestrapporter**

Jens Genberg

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för informationsteknik

Vasa 2014



# EXAMENSARBETE

Författare: Jens Genberg  
Utbildningsprogram och ort: Informationsteknik, Vasa  
Handledare: Ray Pörn

Titel: *Uppdatering av program för utskrift av rutintestrapporter*

---

Datum 2.4.2014      Sidantal 22      Bilagor 1

---

## Abstrakt

Detta examensarbete handlar om uppdateringen av ett program för utskrift av rutintestrapporter och utfördes på uppdrag av ABB Oy, Motors and Generators i Vasa. En ny applikation behövdes för att delar av det ursprungliga programmet var baserat på äldre teknik som inte är kompatibel med Windows 7. Programmets källkod analyserades och ett nytt program konstruerades och testades.

Uppgraderingen innebar en övergång från teknik baserad på ett Word-dokument innehållande Visual Basic for Applications-kod till en fristående .NET-applikation utvecklad med hjälp av C#-programmeringsspråket och ett Windows Presentation Foundation-gränssnitt. Den slutliga produkten är en applikation som söker upp information lagrad i en databas, utför beräkningar och slutligen genererar en rutintestrapport i form av ett formaterat Word-dokument som är färdigt att skrivas ut.

---

Språk: svenska      Nyckelord: .NET, VBA, C#

---

# OPINNÄYTETYÖ

Tekijä: Jens Genberg  
Koulutusohjelma ja paikkakunta: Tietotekniikka, Vaasa  
Ohjaaja: Ray Pörn

Nimike: *Rutiinitestiraporttien tulostamisohjelman päivitys*

---

Päivämäärä 2.4.2014      Sivumäärä 22      Liitteet 1

---

## Tiivistelmä

Tämä opinnäytetyö käsittelee rutiinitestiraporttien tulostamisohjelman päivitystä ja suoritettiin Vaasassa sijaitsevan ABB Oy, Motors and Generators-yksikön toimeksiannosta. Uutta sovellusta tarvittiin kun alkuperäinen ohjelma perustui osittain vanhempaan Windows 7:n kanssa yhteensopimattomaan tekniikkaan. Ohjelman lähdekoodi tutkittiin ja uusi ohjelma kehitettiin ja testattiin.

Päivitys tarkoitti siirtymistä Visual Basic for Applications-koodia sisältävästä Word-dokumenttipohjaisesta tekniikasta erilliseen C#-ohjelmointikielellä ja Windows Presentation Foundation-käyttöliittymällä kehitettyyn .NET-sovellukseen. Lopputulos on sovellus, joka hakee tietokannasta tietoja, suorittaa laskelmia ja lopuksi luo rutiinitestiraportin tulostettavassa Word-dokumenttimuodossa.

---

Kieli: ruotsi      Avainsanat: .NET, VBA, C#

---

# BACHELOR'S THESIS

Author: Jens Genberg

Degree programme and location: Information Technology, Vaasa

Supervisor: Ray Pörn

Title: *Updating a program for printing routine test reports*

---

Date 2.4.2014

Number of pages 22

Appendices 1

---

## Abstract

This thesis concerns the update of a program for printing routine test reports. The work was commissioned by ABB Oy, Motors and Generators in Vaasa. A new application was needed since parts of the original program were based on older technology incompatible with Windows 7. The source code of the program was analyzed and a new program was developed and tested.

The upgrade constituted a transition from technology based on a Word-document containing Visual Basic for Applications-code to a stand-alone .NET-application developed in the C# language with a Windows Presentation Foundation user interface. The final product is an application that looks up information stored in a database, performs calculations and finally generates a routine test report in the form of a printable, formatted Word-document.

---

Language: Swedish

Keywords: .NET, VBA, C#

---

# Innehållsförteckning

<b>1. Inledning .....</b>	<b>1</b>
1.1 Uppdragsgivare .....	1
1.2 Bakgrund.....	1
1.3 Uppdragets omfattning .....	2
<b>2 Teknikbeskrivning .....</b>	<b>3</b>
2.1 Visual Basic for Applications .....	3
2.2 Open Database Connectivity .....	3
2.3 .NET Framework .....	3
2.4 C#.....	4
2.5 Windows Presentation Foundation .....	5
2.6 Windows Communication Foundation .....	6
<b>3 Förverkligande .....</b>	<b>6</b>
3.1 Analys av det ursprungliga programmet.....	6
3.1.1 Duplicerad kod .....	8
3.1.2 Stora klasser .....	8
3.1.3 Överexponering.....	8
3.2 Val av teknik.....	9
3.3 Utveckling av den ersättande applikationen .....	9
3.3.1 Användargränssnittet.....	10
3.3.2 Databehandlingsklassen .....	14
3.3.3 Dokumentbehandlingsklassen.....	16
<b>4 Resultat och diskussion.....</b>	<b>19</b>
4.1 Resultat .....	19
4.2 Diskussion.....	19
4.2.1 Användningen av statiska klasser .....	19
4.2.2 Användargränssnittets responsivitet.....	20
4.2.3 Framtidsutsikter.....	20
<b>Källförteckning.....</b>	<b>21</b>

## Förkortningar

Ord	Förklaring
ERP	Förkortning för Enterprise Resource Planning. Integrerade IT-system som används för ett företags informationshantering.
VBA	Förkortning för programmeringsspråket Visual Basic for Applications.
ODBC	Förkortning för Open Database Connectivity, ett gränssnitt som används för att kommunicera med olika databassystem.
WPF	Förkortning för Windows Presentation Foundation, ett system för att utveckla användargränssnitt.
XAML	Förkortning för Extensible Application Markup Language, språket som används i samband med WPF-utveckling.
WCF	Förkortning för Windows Communication Foundation, ett verktyg som låter applikationer kommunicera med varandra.
COM	Förkortning för Component Object Model, en teknik som låter processer i Windows kommunicera med varandra.

# 1. Inledning

Detta kapitel inleder examensarbetet och börjar med en allmän presentation av uppdragsgivaren. Sedan följer en beskrivning av uppdragets bakgrund samt en förklaring av varför en ny applikation behövdes. Inledningen avslutas med en genomgång av uppdragets omfattning.

## 1.1 Uppdragsgivare

ABB är en multinationell koncern inom elkrafts- och automationsteknologibranschen och grundades år 1988 genom sammanslagningen av svenska Asea och schweiziska Brown Boveri. ABB i Finland har starka rötter i elektroteknikföretaget Strömberg som köptes av Asea år 1987. Koncernen är indelad i följande divisioner:

- Discrete Automation and Motion
- Low Voltage Products
- Process Automation
- Power Systems
- Power Products

Divisionen *Discrete Automation and Motion* arbetar inom områdena frekvensomriktare, energiomvandling, motorer, generatorer och robotar. Till divisionen hör enheten Motors and Generators som är aktiv bland annat i Vasa. Vasas fabrik ansvarar globalt för tillverkningen av lågspänningsmotorer som är säkra att använda i utrymmen där det finns risk för explosioner. [1]

## 1.2 Bakgrund

Uppdragsgivaren hade sedan tidigare ett program för utskrift av rutintestrapporter för motorer och generatorer. Programmet bestod av VBA-kod lagrad i ett Word-dokument som exekverades med hjälp av ett inbyggt formulär. Programmet använde en Oracle-anslutning för att komma åt data från ett ERP-system. Eftersom drivrutinen som användes för Oracle-anslutningen inte var kompatibel med Windows 7 och nyare versioner inte var kompatibla med ERP-systemet, kunde programmet endast köras på datorer med Windows XP installerat. Eftersom alla datorer höll på att flyttas över till Windows 7, krävdes en ny version av programmet för att möjliggöra fortsatt arbete.

En rutintestrapport är ett dokument som innehåller information om en motor eller generator. En kund kan beställa en rutintestrapport i samband med köp av en produkt för att få detaljerad information om produkten. Första delen av rapporten består av information om motorns eller generatorns typ och modell, samt information om kunden och beställningen. Sedan följer teknisk information såsom vilka kombinationer av spänning och nätfrekvens som stöds av produkten samt vilken isoleringsklass den har. Sista delen av rapporten innehåller resultat av tester som utförts på produkten för att mäta bland annat rotationshastighet, effektfaktor och verkningsgrad vid olika spänningar. Även resultat för vibrationstester och upphettningstester anges. I bilaga 1 ges ett exempel på hur en rutintestrapport kan se ut.

### **1.3 Uppdragets omfattning**

Uppdraget som detta examensarbete baseras på är utvecklingen av en ny, fristående applikation med motsvarande funktionalitet som det ursprungliga programmet. Ett krav var att den nya applikationen var kompatibel med Windows 7. Den behövde inte vara kompatibel med någon äldre version av Windows eller något annat operativsystem. Ett annat krav var att applikationen var lätt att använda. Eftersom det fanns flera användare var det viktigt att programmet inte krävde någon komplicerad installationsprocess. Teknik och programmeringsspråk fick väljas fritt och källkoden till det gamla programmet fanns tillgänglig. Även om det ursprungliga programmet och den nya applikationen använder sig av databaser, begränsades uppdraget till programmering av själva applikationen som användaren ser och innehåller inget direkt databasarbete. Databaserna existerade från förr och all direkt databastillgång sker på en server. Databasförfrågningarna som behövdes skrevs av en annan programmerare.



## 2 Teknikbeskrivning

Detta kapitel beskriver vilken teknik som använts vid arbetet med detta lärdomsprov i form av programmeringsspråk, ramverk och verktyg. Först behandlas tekniken som det ursprungliga programmet baserar sig på och sedan behandlas tekniken som valts till förverkligandet av den nya applikationen.

### 2.1 Visual Basic for Applications

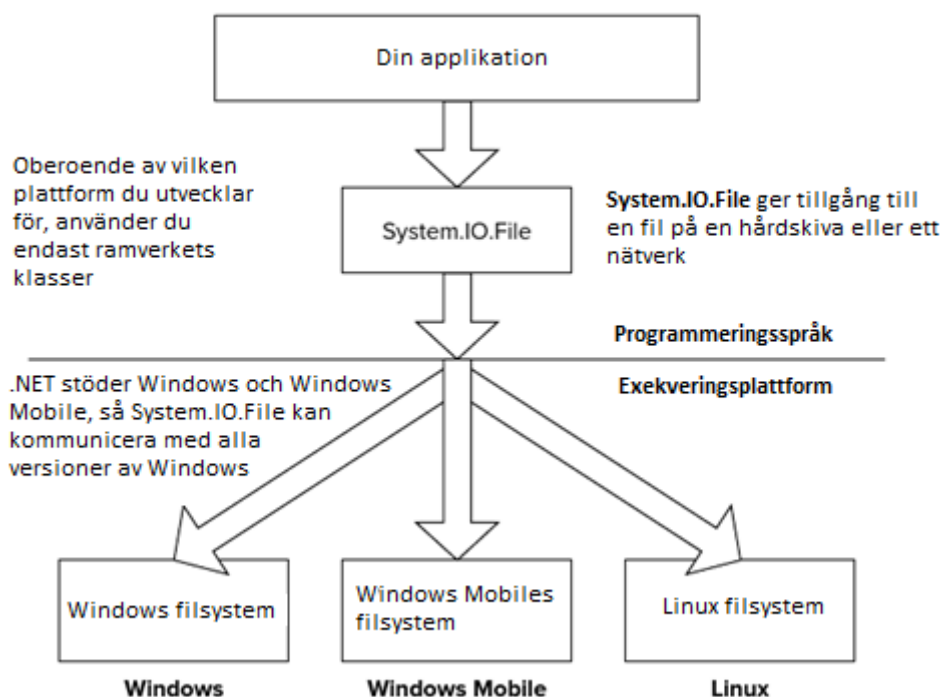
Visual Basic for Applications eller VBA är det gemensamma makro-programmeringsspråket för alla Microsoft Office-program och introducerades i samband med Office Excel 5 år 1993. VBA är ett objektbaserat programmeringsspråk som är identiskt till Visual Basic 6 i sin struktur. Som nämndes i kapitel 1.2 är den ursprungliga applikationen utvecklad med denna teknik. [2]

### 2.2 Open Database Connectivity

Open Database Connectivity eller ODBC är ett gränssnitt utvecklat av Microsoft som används i applikationer för att få tillgång till databaser av olika typer. Tre komponenter krävs för användning av ODBC. Dessa är en ODBC-klient, en ODBC-drivrutin och en databashanterare. ODBC-klienten är i det ursprungliga programmet fall VBA-applikationen som visar data för användaren. ODBC-drivrutinen är till för att låta klienten kommunicera med databashanteraren. I det ursprungliga programmet fall användes en Oracle-baserad databashanterare och således krävdes en ODBC-drivrutin för Oracle. [3]

### 2.3 .NET Framework

År 2002 lanserades .NET Framework av Microsoft. Detta ramverk är ett objektorienterat abstraktionslager som ligger mellan applikationen och operativsystemet. Ramverket låter applikationer kommunicera med operativsystemets olika delsystem, såsom systemen för filhantering, nätverkskommunikation och grafik med hjälp av ett antal inbyggda klasser. [4]. Detta underlättar programmering, eftersom funktionaliteten är den samma oberoende av vilket programmeringsspråk som används. Ramverket används främst i Windows än så länge, men det är möjligt att utvidga till andra operativsystem. Mono-projektet [5] har släppt versioner för bland annat Mac OS X, Linux och Android. Figur 1 visar ett exempel på hur en av ramverkets klasser används. Applikationen i figur 1 använder sig av .NET-klassen *System.IO.File* för att få tillgång till filsystemet. Figur 1 förklarar hur samma klass används oberoende om man arbetar mot olika versioner av Windows eller mot Linux, trots att operativsystemen använder sig av olika filsystem.



Figur 1. Ett exempel på kommunikation via .NET Framework.

## 2.4 C#

C# är ett programmeringsspråk utvecklat av Microsoft som från början var tänkt att användas tillsammans med .NET Framework samt stöda de senaste principerna för objektorienterad programmering [6]. Språket är till stor del härlett från C och C++. Tillsammans med Visual Basic är C# det vanligaste språket vid .NET-programmering. De båda språken ger i allmänhet tillgång till samma funktionalitet. Figur 2 visar samma algoritm utförd med båda programmeringsspråken. Visual Basic använder sig mera av nyckelord på engelska, medan C# använder symboler såsom parenteser och klamrar och påminner mera om C/C++.

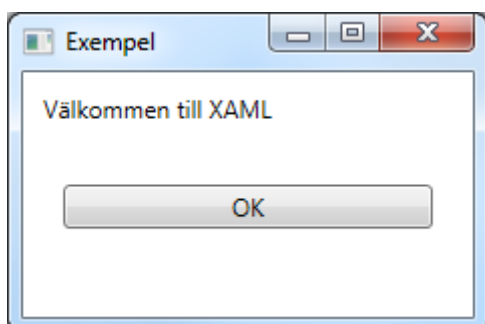
C#	Visual Basic .NET
<pre>for (int i = 0; i &lt; tal; ++i) { // gör något } // kör loopen från noll upp till ett mindre än det givna talet</pre>	<pre>For i As Integer = 0 To tal - 1 ' gör något Next ' kör loopen från noll upp till ett mindre än det givna talet</pre>

Figur 2. Ett exempel på skillnaden mellan syntaxen i C# och Visual Basic .NET.

## 2.5 Windows Presentation Foundation

Windows Presentation Foundation eller WPF är en del av .NET Framework och används för att skapa användargränssnitt och annan grafik för applikationer. Till skillnad från äldre teknologier såsom Windows Forms, använder sig WPF av DirectX, vilket tillåter mera avancerade effekter samt bättre prestanda med användning av hårdvaruacceleration. En annan viktig egenskap som introducerades med WPF är Extensible Application Markup Language eller XAML. XAML är ett märkspråk som används för att definiera användargränssnitt till en .NET-applikation, och separera det från koden. Detta gör det lättare att dela upp utvecklingsarbetet mellan programmerare och GUI-designers. Figur 3 visar hur samma användargränssnitt skapas med C#-kod respektive XAML. WPF-gränssnitt är uppbyggda av så kallade kontroller som används för att visa information och låta användaren mata in information. Kontroller av typerna *Stackpanel*, *TextBlock* och *Button* används i exemplet i figur 3. XAML-metoden kräver ofta mindre text och resultatet syns direkt utan att man behöver köra programmet om man använder Microsoft Visual Studio. [7]

C#	XAML
<pre>StackPanel stackPanel = new StackPanel(); this.Content = stackPanel;  TextBlock textBlock = new TextBlock(); textBlock.Margin = new Thickness(10); textBlock.Text = "Välkommen till XAML"; stackPanel.Children.Add(textBlock);  Button button = new Button(); button.Margin = new Thickness(20); button.Content = "OK"; stackPanel.Children.Add(button);</pre>	<pre>&lt;StackPanel&gt;  &lt;TextBlock Margin="10"&gt; Välkommen till XAML &lt;/TextBlock&gt;  &lt;Button Margin="20"&gt; OK &lt;/Button&gt;  &lt;/StackPanel&gt;</pre>



Figur 3. C#-kod och XAML används för att rita upp samma användargränssnitt.

## 2.6 Windows Communication Foundation

Windows Communication Foundation, eller WCF, hör också till .NET Framework och används för att bygga serviceorienterade applikationer. WCF kombinerar funktionaliteten i flera äldre teknologier, såsom Web Services, .NET Remoting, Enterprise Services och Message Queuing. Med hjälp av WCF kan man till exempel konfigurera en tjänst på en server som sedan kan anropas av en klientapplikation för att få tillgång till data på servern. Hur detta fungerar beskrivs närmare i kapitel 3.3.2.

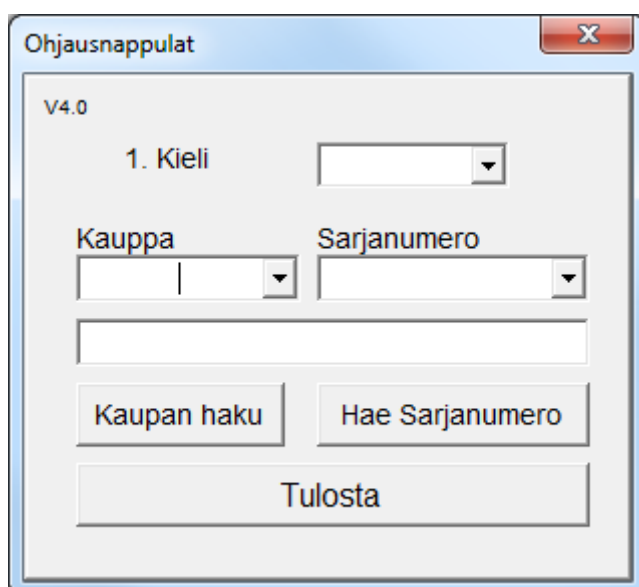
## 3 Förverkligande

Detta kapitel beskriver examensarbetets förverkligande, det vill säga planeringen och utvecklingen av applikationen. Detta innefattar en analys av koden i det ursprungliga programmet samt motiveringar för val av teknik för utvecklingen av den ersättande applikationen. I slutet av kapitlet ges exempel på andra teknikalternativ som kunde ha använts i den ersättande applikationen.

### 3.1 Analys av det ursprungliga programmet

Jag påpekade inledningsvis att jag hade tillgång till källkoden för programmet mitt uppdrag baserade sig på. Första steget jag tog i utvecklingen av den nya applikationen var att analysera den gamla koden för att sedan kunna återskapa funktionaliteten. Liksom jag nämnde i kapitel 1.2 existerar den gamla applikationen i ett Word-dokument. Källkoden gick att läsa i Microsoft Words inbyggda Visual Basic-editor. Nedan följer en beskrivning av hur programmet används i stora drag.

För att starta programmet öppnar användaren Word-dokumentet, varpå en kodsnuitt körs för att rita upp användargränssnittet. Användaren kan då välja önskat språk för rutintestrapporterna samt skriva ut rapporter för alla motorer i en order enligt ordernummer eller för enskilda motorer enligt serienummer. Inmatning av ordernummer eller serienummer sker via *combobox*-kontroller och så fort man förflyttar sig från kontrollen uppdateras den andra med tillhörande information. Om man fyller i ett existerande ordernummer, fylls *combobox*-kontrollen för serienummer automatiskt med de tillhörande serienumren och om man fyller i ett existerande serienummer, läggs det tillhörande ordernumret till *combobox*-kontrollen för ordernummer. För att söka på det valda ordernumret eller serienumret används motsvarande sökknapp. Figur 4 visar hur gränssnittet ser ut. Det innehåller kontroller för val av språk, inmatning av ordernummer eller serienummer, sökning på ordernummer eller serienummer samt utskrift.



Figur 4. Gränssnittet för det ursprungliga programmet.

Då användaren fyllt i en av combobox-kontrollerna öppnas en databasanslutning av typen ODBC. Detta sker automatiskt genom en så kallad *change*-händelse som är kopplad till kontrollen via koden. Om det är ordernumret som fyllts i, körs en förfrågan mot databasen som hämtar alla tillhörande serienummer. Dessa lagras programmet i en lista. Om serienumret istället har fyllts i, hämtas tillhörande ordernummer samt ett certifikatnummer för hela ordern.

Om användaren väljer att utföra en sökning enligt serienummer, söker programmet upp en Word-dokumentmall i samma mapp som programmet körs från och öppnar den. Om den redan är öppen, stängs den och öppnas på nytt. Sedan raderas all eventuell text från malldokumentet. Först används en databasförfrågan för att fylla i så kallad header-information i huvuddokumentet. Denna förfrågan görs på basen av ordernumret som programmet redan har hämtat efter att användaren fyllt i serienumret. Sedan kontrollerar programmet att det ifyllda serienumret existerar i databasen. Om det existerar körs en funktion som hämtar mätresultat och annan info enligt serienumret samt fyller i denna information i huvuddokumentet. Efter att den funktionen körts färdigt, kopieras huvuddokumentets innehåll till den tomma dokumentmallen. Slutligen visas en dialog för att spara dokumentet med ett namn baserat på serienumret.

Om användaren väljer att utföra en sökning enligt ordernummer, görs samma kontroll av dokumentmallen. Ifall ordernumret hittas i databasen, fylls header-informationen i. Sedan fylls informationen för varje tillhörande serienummer in i huvuddokumentet en sida i gången och kopieras till dokumentmallen i en programslinga. Slutligen visas en dialog för att spara dokumentet som nu innehåller en testrapport per sida enligt serienummer. Dokumentet får ett namn baserat på ordernumret.

Då jag undersökte VBA-koden, upptäckte jag flera typer av vad som först definierades som *code smells* i [8]. Dessa är symptom på bristfällig design eller arkitektur hos ett program. Jag kommer här att ange vilka typer av dessa problem jag hittade i koden.

### 3.1.1 Duplicerad kod

Koden innehåller duplicering, det vill säga kod som upprepas på flera ställen. Till exempel upprepas koden som kontrollerar om dokumentmallen redan är öppen. Även kod som används för att analysera ordernummer och plocka ut information om ordern förekommer på flera ställen. Enligt [8] bör kodsnuttar som används i flera skeden av programmet sättas in i separata metoder som sedan kan anropas. Man minskar risken för misstag vid uppdatering av koden om samma kod endast behöver uppdateras på ett ställe. Även enligt [9] anses duplicering göra kod svårare att förstå och underhålla. Författarna talar om *DRY*-principen där *DRY* står för *Don't Repeat Yourself*. Ett liknande tecken på bristande planering är att programmet använder två olika sätt för att sätta in data i tabellen som finns i Word-dokumentet. På vissa ställen används en inbyggd VBA-funktion och på andra ställen används en metod som anropar samma funktion.

### 3.1.2 Stora klasser

Trots att VBA-editorn tillåter uppdelning av kod i klasser eller moduler, har allting skrivits i en och samma fil bakom användargränssnittet. Enligt [8] är kod lättare att förstå ifall den är uppdelad i ett antal klasser med tydliga ansvarsområden. Med en sådan struktur blir det också lättare att upptäcka andra problem, såsom duplicerad kod.

### 3.1.3 Överexponering

Enligt [10] bör man inte överexponera data genom att göra variabler onödigt synliga, t.ex. genom att göra dem publika. Detta gör det svårare att kontrollera beroenden i koden, och orsakar lätt namngivningskonflikter ifall man behöver liknande variabler på flera ställen. Det ursprungliga programmets kod innehåller flera exempel på detta. Bland annat har osynliga *textbox*-kontroller i användargränssnittet använts för att lagra information, för att den ska vara tillgänglig globalt, istället för att använda mera begränsad synlighet. Detta gjorde koden svårare att förstå, eftersom det inte var klart var man skulle hitta definitionen på en variabel som används i koden. Programmeraren har också i flera fall definierat två variabler som har samma namn – den ena variabeln på engelska och den andra på finska. Detta gjorde också koden svårare att förstå och är ett bra exempel på en namngivningskonflikt som orsakats av överexponering.

## 3.2 Val av teknik

Jag valde att skapa en helt ny, fristående applikation programmerad i C# med ett användargränssnitt utvecklat i Windows Presentation Foundation. Det var självklart för mig att använda ett språk som stöds av .NET Framework, eftersom ramverket innehåller inbyggda klasser för att kommunicera med och kontrollera instanser av Microsoft Office-program. Microsoft Word finns tillgängligt på alla uppdragsgivarens datorer och används även av det ursprungliga programmet för att skapa rapporterna. Jag valde att skriva koden i C# eftersom jag har mest erfarenhet av programmering i det språket. Av samma anledning valde jag att konstruera användargränssnittet med hjälp av WPF. Dessa tekniker presenteras i teknikbeskrivningen i kapitel 2.

Som programmeringsspråk skulle Visual Basic ha låtit mig använda precis samma .NET-klasser och även WPF för att utveckla en väldigt liknande applikation. Enda orsaken till att jag valde C# var att jag var mera bekant med den syntaxen.

Jag kunde ha använt Windows Forms istället för WPF vid utvecklingen av användargränssnittet, men bestämde mig ändå för att WPF var ett bättre val på grund av möjligheten att använda XAML. Det lät mig separera användargränssnittets design från resten av koden, vilket behandlades i kapitel 2.4.

Hade jag varit mera bekant med C++, kunde jag ha undersökt det alternativet noggrannare. C++ stöder flera .NET-funktioner och det är även möjligt att använda Microsoft Foundation Class-biblioteket för att utveckla enkla gränssnitt för Windows-applikationer.

Jag har även erfarenhet av programmering i Java och att programmera applikationen i Java kunde ha varit ett bra alternativ ifall det hade funnits krav på att applikationen skulle kunna köras på olika plattformar. I det här fallet var dock .NET en betydligt smidigare lösning.

[11, 12, 13]

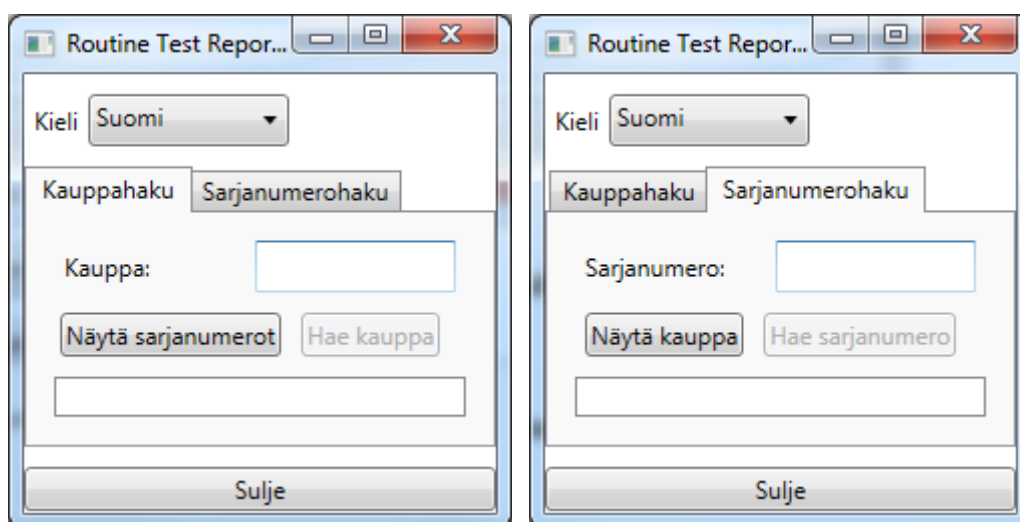
## 3.3 Utveckling av den ersättande applikationen

Jag valde att basera den nya applikationen på två klasser – en databehandlingsklass och en dokumentbehandlingsklass. Databehandlingsklassen används för att hämta information tillhörande beställningar och enskilda produkter, samt för att utföra uträkningar på och formatera denna information. Dokumentbehandlingsklassen används för att skapa testrapporter genom att fylla i informationen som genererats av databehandlingsklassen i ett Word-dokument. Utöver dessa två klasser, innehåller applikationen ett grafiskt användargränssnitt samt kod för att sammankoppla gränssnittet med funktionaliteten i klasserna.

Jag gjorde databehandlingsklassen och dokumentbehandlingsklassen till statiska klasser. Detta betyder att klasserna inte kan instansieras till objekt, utan de används genom att man hänvisar till klassens namn. Jag ansåg att statiska klasser var lämpliga för dessa uppgifter, eftersom applikationen endast behöver hantera en databasanslutning åt gången samt ett dokument åt gången. [14]

### 3.3.1 Användargränssnittet

Applikationens slutgiltiga användargränssnitt syns i figur 5. Gränssnittet är uppbyggt med hjälp av WPF-kontroller av typerna *TabControl*, *Button*, *TextBox*, *ComboBox*, *ListBox* och *ProgressBar*. *TabControl*-kontrollen används för dela upp gränssnittet i två flikar och avskilja de två olika typerna av sökningar. En *ComboBox*-kontroll används för att välja språk. *TextBox*-kontrollerna och *Button*-kontrollerna används för att utföra sökningar. *ListBox*-kontrollen används för att visa tillhörande serienummer respektive ordernummer. Längst ned på fönstret finns en lägesindikator.



Figur 5. Den nya applikationens gränssnitt.

Vid utformningen av gränssnittet strävade jag till att göra det så likt det ursprungliga programmet gränssnitt som möjligt enligt uppdragsgivarens krav. Man kan säga att applikationen har tre huvudsakliga användningsfall. Dessa är programstart, sökning enligt ordernummer och sökning enligt serienummer. Nedan beskrivs vad som händer i användningsfallen.



### ***Programstart***

Det första som händer då applikationen startas är att en initieringsmetod i databehandlingsklassens körs. Denna metod ser till att en anslutning till databashanteraren som används för sökningarna öppnas. Detta åstadkoms med hjälp av en så kallad *Service Reference*. Även två objekt av typen *BackgroundWorker* skapas och konfigureras. Begreppen *Service Reference* och *BackgroundWorker* förklaras senare i kapitlet.

### ***Sökning på ordernummer***

Vid sökning på ordernummer används den första fliken av användargränssnittet. Användaren fyller i ett ordernummer i textrutan och klickar på en av de två knapparna. Den första knappen används för att söka upp alla serienummer som tillhör ordern ifråga och lägga dem i listan nedanför knapparna. Om användaren sedan klickar på ett av serienumren i listan, öppnas den andra fliken och serienumret kopieras till den flikens textruta. På det sättet kan användaren skriva ut en testrapport för en enskild serienummer, ifall endast ordernumret är tillgängligt. Den andra knappen används för att söka upp information om alla tillhörande serienummer och skapa en testrapport utifrån dem.

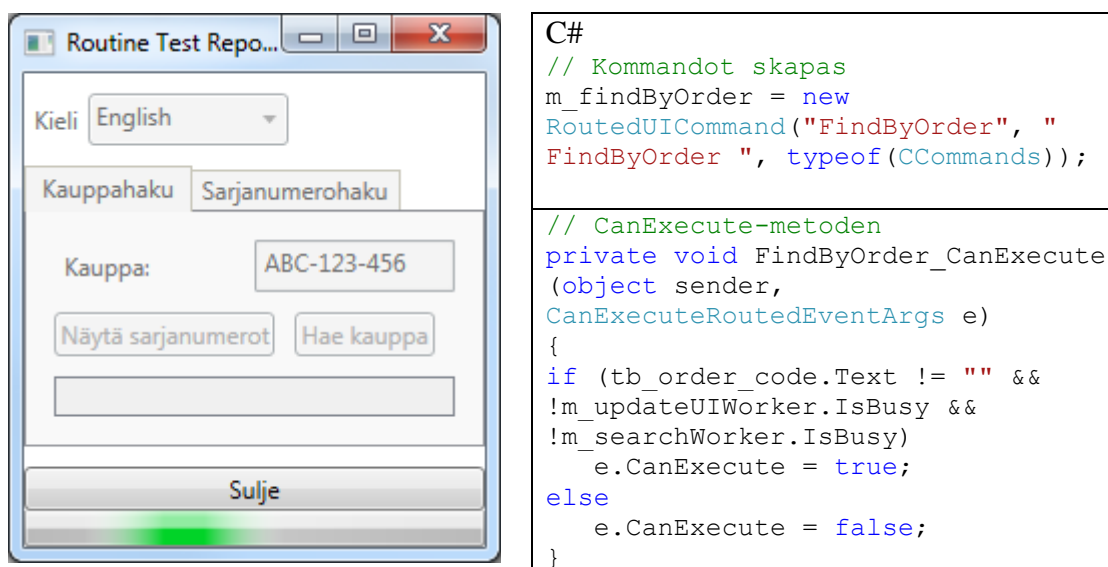
### ***Sökning på serienummer***

Den andra fliken används för att söka på enskilda serienummer. Användaren fyller i ett serienummer i textrutan och klickar på en av de två knapparna. I likhet med den första fliken, kan användaren även här välja att söka fram det tillhörande ordernumret, ifall endast ett serienummer är tillgängligt. På det viset kan användaren skriva ut andra testrapporter från samma order. Användaren kan också söka upp information om det inmatade serienumret och skapa en testrapport endast utifrån det.

Två olika programmeringstekniker – *Commanding* och *BackgroundWorker* – har använts vid utvecklingen av applikationen för att få användargränssnittet att kommunicera med de underliggande klasserna för databehandling och dokumentbehandling. Dessa beskrivs i följande stycke.

### *Commanding*

*Commanding* är en del av WPF och kan användas för att hantera kopplingen mellan kontroller i användargränssnittet och funktioner i koden. Med *Commanding* kan man också inaktivera kontroller, ifall rätt villkor inte uppfylls för deras användning. Till skillnad från användningen av en enkel *onClick*-händelse för att koppla ihop en XAML-kontroll med en bakomliggande metod, delas koden upp i två metoder. *CanExecute*-metoden innehåller kod för att kontrollera om kommandots villkor är uppfyllda, medan *Executed*-metoden innehåller koden som ska köras, ifall villkoren uppfylls. I rutintestapplikationen används *CanExecute*-metoden för att förhindra användaren från att starta en sökning, ifall textfältet är tomt eller en sökning redan pågår. Figur 6 demonstrerar användargränssnittet med inaktiverade kontroller samt en aktiv lägesindikator som visar att applikationen är upptagen. Även koden som skapar kommandot samt kommandots *CanExecute*-metod visas. Figur 7 innehåller koden för kommandots *Executed*-metod samt XAML-kod som binder sökknappen till kommandot. [15]



Figur 6. Användargränssnittet då applikationen är upptagen samt koden för att skapa kommandot och kommandots *CanExecute*-metod.

<pre> C# // Executed-metoden private void FindByOrder_Executed(object sender, ExecutedRoutedEventArgs e) {     string[] arguments = { "findByOrder",    tb_order_code.Text };     m_searchWorker.RunWorkerAsync(arguments); } </pre>
<pre> XAML // Söknappen binds till kommandot &lt;CommandBinding Command="local:CCommands.FindByOrder" CanExecute=" FindByOrder_CanExecute" Executed=" FindByOrder_Executed"&gt;  &lt;Button Name="bt_findOrder" Command="local:CCommands.FindByOrder"&gt;Hae kauppa&lt;/Button&gt; </pre>

Figur 7. C#-koden för kommandots Executed-metod för sökning på order samt XAML-koden som binder söknappen till kommandot.

### **Backgroundworker**

*BackgroundWorker* är en .NET-klass som används för att köra tidskrävande operationer på en separat tråd, i bakgrunden, utan att användargränssnittet låser sig. Rutintestapplikationen använder sig av *BackgroundWorker*-objekt för all databastillgång. Eftersom den tidskrävande koden körs asynkront på en annan tråd, kan användargränssnittet uppdateras och visa en lägesindikator. Figur 7 ovan visar hur *BackgroundWorker*-tråden för sökning startas i *FindByOrder*-kommandots *Executed*-metod. Figur 6 ovan visar även lägesindikatorn som syns längst nere i fönstret då applikationen är upptagen. I figur 8 syns koden som körs på *BackgroundWorker*-tråden vid sökning på serienummer eller ordernummer. Det är *BackgroundWorker*-klassens *ReportProgress*-metod som tillåter att meddelanden skickas till användargränssnittet fast det körs på en separat tråd. [16]

En annan möjlighet hade varit att använda klasserna i namnrymden *System.Threading* till att skapa en separat tråd manuellt. Jag valde dock att använda mig av *BackgroundWorker*, för att spara på tid och arbete, eftersom det tillåter användningen av färdiga händelser för att kommunicera med huvudtråden. [17]

```

C#
void m_searchWorker_DoWork(object sender, DoWorkEventArgs e)
{
    m_searchWorker.ReportProgress(0);
    string[] arguments = (string[])e.Argument;

    if (arguments[0] == "findBySerial")
    {
        CDocumentHandling.FindBySerial(arguments[1]);
    }
    else if (arguments[0] == "findByOrder")
    {
        CDocumentHandling.FindByOrder(arguments[1]);
    }
    m_searchWorker.ReportProgress(100);
}

```

Figur 8. DoWork-metoden för BackgroundWorker-objektet som används vid sökning på serienummer eller ordernummer.

### 3.3.2 Databehandlingsklassen

Databehandlingsklassen ansvarar för att hämta information för testrapporterna från databashanteraren. Till detta används en *Service Reference* som beskrivs senare i kapitlet. Till klassen hör också metoder som används för att strukturera informationen till ett mera lätthanterligt format samt metoder som används för att göra uträkningar på basen av mätvärden som hämtats. Figur 9 visar *RatingPlateQuery3*, en av metoderna som används för att hämta information om en motor eller generator. Metoden använder sig av *GenericARRAY* som är en klass som definierats på serversidan för att lagra ett enskilt resultat från en databasförfrågan. Metoden *getGenericData* på serversidan kan returnera ett eller flera sådana resultat. *GetGenericData* används för alla sökningar och tar som argument namnet på databasförfrågan, en array med parameternamn samt en array med parametervärden. Databasförfrågan körs sedan på servern, som returnerar resultatet till klienten.

```

C#
public static GenericARRAY[] RatingPlateQuery3(string salesperson,
string order, int position){
    var paramNames = new string[3];
    paramNames[0] = "Myyja";
    paramNames[1] = "Kauppa";
    paramNames[2] = "Positio";

    var paramValues = new string[3];
    paramValues[0] = salesperson;
    paramValues[1] = order;
    paramValues[2] = position.ToString();

    GenericARRAY[] result = m_client.getGenericData("Ratingplate3",
paramNames, paramValues);

    return result;}

```

Figur 9. En av metoderna för att hämta data från servern

Figur 10 visar en del av klassen *GenericARRAY* som är definierad på servern och skriven i Java. Klassen innehåller en strängvektor med längden 100 som lagrar namnen på fält som returneras. Själva värdena lagras i en strängvektor om de är i textform eller i en vektor av typen *Double* om de är numeriska. Detta gör att klassen kan användas för att returnera olika typer av data som resulterar från olika databasförfrågningar.

```

Java
public class GenericARRAY
{
    private String[] _Def;
    private String[] _S;
    private Double[] _N;

    public GenericARRAY()
    {
        _Def = new String[100];
        _S = new String[100];
        _N = new Double[100];
    }

    public String[] getDef() {
        return _Def;
    }

    public void setDef(String def, int index) {
        _Def[index] = def;
    }

    .....
}
}

```

Figur 10. Klassen *GenericARRAY* som används för att returnera sökresultat från servern till klienten.

### ***Service Reference***

För att kommunicera med WCF-tjänster från en .NET-applikation används någonting som kallas servicereferenser. Det som krävs för att skapa en anslutning till en tjänst från en .NET-applikation är adressen till en WCF-server som fylls i en dialog i Visual Studio. Visual Studio genererar då automatiskt en så kallad proxy-klass. Denna klass kan man sedan instansiera i koden och använda. I rutintestapplikationen använde jag mig av en sådan proxy-klass för att få tillgång till en databashanterare som finns på en lokal server hos uppdragsgivaren. Efter att en instans av klassen skapats, kunde jag från applikationen anropa alla funktioner som definierats för tjänsten ifråga på serversidan. På det här viset kan rutintestapplikationen hämta mätdata för motorer och annan information som behövs för att skapa en rutintestrapport. [18]

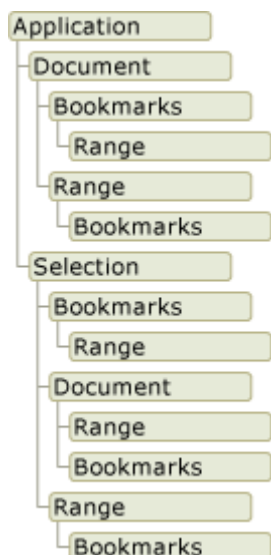
### **3.3.3 Dokumentbehandlingsklassen**

Dokumentbehandlingsklassen ansvarar för att skapa själva rutintestrapporten utifrån informationen som färdigställts av databehandlingsklassen. Till detta hör skapandet av nya dokument, insättningen av rubriker och fältnamn enligt det valda språket samt sparande av dessa dokument. Detta åstadkoms med hjälp av API-objekt för Microsoft Office, vilket innebär att man inifrån applikationen kan starta instanser av t.ex. Microsoft Word samt skapa och editera Word-dokument. Jag hade störst nytta av .NET-ramverket i samband med dokumentbehandlingsklassen, eftersom tillgången till Microsoft Word gjorde att jag inte behövde uppfinna något eget sätt att hantera och spara dokument.

### ***Interoperabilitet med Microsoft Office***

För att få tillgång till Microsoft Word i en .NET-applikation krävs att man sätter in en referens till *Microsoft.Office.Interop.Word*. Detta är en namnrymd som innehåller alla klasser som behövs för att kommunicera med Word via någonting som kallas *Word Object Model*. Med hjälp av denna modell kan man i koden skapa nya instanser av Microsoft Word och automatisera dem. Man kan med andra ord få en applikation att manipulera Word-dokument på samma sätt som vilken vanlig användare som helst.

I mitt fall har jag i huvudsak använt mig av tre av objekten i *Word Object Model*. Dessa är *Application*, *Document* och *Selection*. *Application*-objektet representerar själva instansen av Word-programmet och existerar högst uppe i hierarkin. Det innehåller bland annat metoder för att skapa och öppna dokument. *Document*-objektet representerar ett enskilt dokument som är öppet i Word-instansen man skapat. Det används för att få tillgång till och editera dokumentets innehåll bland annat via *Selection*-objektet. *Selection*-objektet anger vilken del av dokumentet som är markerat och var kursorn är, vilket påverkar var insättningar, ändringar och raderingar av text görs. Figur 11 visar modellens hierarki.



Figur 11. En översikt över hur Word Object Model är strukturerad. [19]

Användningen av Microsoft Office-automation underlättar då man behöver skapa ett stort antal liknande dokument eller utföra liknande ändringar på ett stort antal dokument. I figur 12 visas hur enkelt det är att öppna ett Word-dokument från en .NET-applikation och sätta in text. Metoden *Init* används för att initiera Word-applikationen. Där innebär *WdAlertLevel.wdAlertsNone* att Word-instansen inte framkallar dialoger som i onödan skulle kunna stjäla fokus från applikationens gränssnitt. Metoden *InsertIntoCell* används för att sätta in text i olika delar i tabellen som finns i rutintestrapporten. Något som bör noteras är att Microsoft i detta fall använder ett-baserad indexering, även om C# i allmänhet använder noll-baserad indexering. Detta påverkar hur man hänvisar till kolumner och rader i koden.

```

C#
public static void Init()
{
    // Den här metoden initierar Word-applikationen
    m_word = new Application();
    m_word.DisplayAlerts = WdAlertLevel.wdAlertsNone;
    // Öppnar mallen samt ett tomt Word-dokument
    m_templateDoc = m_word.Documents.Open(@"\RUTIINIPK04.doc",
    ReadOnly: true);
    m_emptyDoc = m_word.Documents.Open(@"\EMPTY.doc",
    ReadOnly: true);
}

public static void InsertIntoCell(int row, int column, string text)
{
    // Den här metoden sätter in text i en specificerad
    m_templateDoc.Tables[1].Cell(row, column).Range.Text = text;
}
  
```

Figur 12. Kod för att skapa en instans av Word, öppna ett dokument samt lägga till text. Dokumentet är strukturerat så att all information sätts in i en tabell.

## Minneshantering

Det största problemet jag stötte på vid Microsoft Office-automationen var Word-instansernas tendens att hänga kvar i minnet efter att applikationen hade avslutats. Efter att ha använt applikationen ett tag under testningen var det vanligt att jag hade flera processer som fortfarande var igång i bakgrunden. Detta förhindrade inte att nya sökningar gjordes, men det kunde ha lett till allvarliga minnesläckor i längden. Figur 13 visar hur jag löste problemet med hjälp av en metod. Denna metod körs före en ny sökning startas och då applikationen stängs. Koden ser till att frigöra alla COM-objekt som används för att kommunicera med Word-instansen och fungerar oberoende om användaren redan har stängt Word-dokumentet manuellt eller inte. [20]

```
C#
public static void CollectGarbage()
{
    // Den här metoden förhindrar att Word-processer hänger kvar i minnet
    if (m_word != null)
    {
        if (m_templateDoc != null)
        {
            m_templateDoc.Close(SaveChanges:
                WdSaveOptions.wdDoNotSaveChanges);
            Marshal.FinalReleaseComObject(m_templateDoc);
            m_templateDoc = null;
        }
        if (m_emptyDoc != null)
        {
            m_emptyDoc.Close(SaveChanges: WdSaveOptions.wdDoNotSaveChanges);
            Marshal.FinalReleaseComObject(m_emptyDoc);
            m_emptyDoc = null;
        }
        m_word.Quit(SaveChanges: WdSaveOptions.wdDoNotSaveChanges);
        Marshal.FinalReleaseComObject(m_word);
        m_word = null;
    }
    GC.Collect();
    GC.WaitForPendingFinalizers();
}
```

Figur 13. En metod som ser till att alla Word-processer avslutas då de inte längre behövs.



## 4 Resultat och diskussion

Detta kapitel behandlar examensarbetets resultat samt innehåller diskussion om den resulterande applikationen och hur vissa delar av utvecklingen kunde ha förverkligats annorlunda. Kapitlet avslutas med tankar kring hur applikationens framtid kan komma att se ut.

### 4.1 Resultat

Resultatet av examensarbetet är en fristående applikation som används av 5–10 personer på enheten ABB Motors and Generators i Vasa. Applikationen har samma egenskaper som det äldre programmet det ersätter samt används på ett liknande sätt. Applikationen går enkelt att kopiera till en ny dator och fungerar med operativsystemet Windows 7. Således uppfyller applikationen alla uppdragsgivarens krav och löser problemet med inkompatibilitet som det ursprungliga programmet hade.

### 4.2 Diskussion

Efter att ha färdigställt applikationen har jag kommit fram till att det finns två områden som kunde förbättras och som jag skulle förverkliga annorlunda ifall jag fick göra om utvecklingsprocessen. Dessa är användningen av statiska klasser, vilket främst påverkar underhåll av koden och användargränssnittets responsivitet, vilket även kan påverka användarupplevelsen.

#### 4.2.1 Användningen av statiska klasser

Valet att basera en stor del av applikationen på statiska klasser kan ha orsakat flera problem än vad det löste. De två huvudklasserna där största delen av programkoden finns är dokumentbehandlingsklassen och databehandlingsklassen och de är ansvariga för kommunikation med databasen respektive ifyllande av Word-dokumentet, vilket beskrevs i kapitlen 3.3.2 och 3.3.3. Enligt Martin Rybaks blogginlägg [21] kan användningen av statisk kod orsaka flera problem. Det påpekas att koden lätt kan bli svårförståelig, svårtestad och svår att arbeta med, vilket också visade sig vara sant i det här fallet. Om dokumenthanteringen hade förverkligats med hjälp av instansierade klasser istället för statiska, hade det i min mening underlättat frigörandet av COM-objekten som används vid kommunikation med Word. Resurserna kunde ha frigjorts direkt då instansens destruktör kördes efter att ett dokument skrivits klart, vilket hade förenklat debugging-processen. Valet att använda statiska klasser påverkade ändå inte applikationens funktionalitet i slutändan och är ingenting som skadar användarupplevelsen.

### 4.2.2 Användargränssnittets responsivitet

Eftersom applikationens användargränssnitt innehåller en lägesindikator, får användaren viss feedback medan applikationen arbetar. Denna lägesindikator visar däremot inte hur mycket arbete som är gjort och hur mycket som finns kvar. Detta hade jag kunnat lösa bättre genom användningen av *Threads* istället för *Backgroundworker*, vilket hade låtit mig använda en lägesindikator för att mera exakt ange hur lång tid man måste vänta innan en sökning är klar. Kombinationen av en databasanslutning som inte är särskilt snabb och bristande feedback till användaren kan ge denne en negativ upplevelse av applikationen. Eftersom jag inte hade någon kontroll över databasanslutningen och dess prestanda kunde jag ha fokuserat på att utveckla ett mera responsivt gränssnitt.

### 4.2.3 Framtidsutsikter


Som det ser ut kommer den nya rutintestapplikationen att användas hos uppdragsgivaren i fortsättningen. Den gamla versionen av programmet används inte längre, eftersom inga arbetsstationer i dagens läge har stöd för den. Det existerar även ett annat liknande program för utskrift av rutintestrapporter lagrade med annan databasteknik som fortfarande är i användning. Det skulle i min mening vara möjligt att bygga på rutintestapplikationen som detta examensarbete resulterade i, för att integrera även detta programs funktionalitet i en och samma applikation.

## Källförteckning

- [1] *ABB lyhyesti* <http://new.abb.com/fi/abb-lyhyesti/> (hämtat: 17.3.2014)
- [2] Kimmel, Paul T. Bullen, Stephen & Green, John. (2004) *Excel 2003 VBA Programmer's Reference*. Hoboken, NJ, USA: Wiley. Sid xxiv.
- [3] *ODBC - Open Database Connectivity Overview*. Microsoft Support. <http://support.microsoft.com/kb/110093> (hämtat: 31.12.2013)
- [4] Newsome, Bryan. (2012). *Beginning Visual Basic 2012*. Somerset, NJ, USA: Wiley. Sid 31.
- [5] *Mono* <http://www.mono-project.com> (hämtad 4.1.2014)
- [6] Nagel, Christian. Evjen, Bill. Glynn, Jay. Watson, Karli. Skinner, Morgan. (2012). *Professional C# 2012 and .NET 4.5*. Somerset, NJ, USA: Wiley. Sid 4.
- [7] MacDonald, Matthew. (2010) *Pro WPF in C# 2010*. Apress. Sid 1-2, 5.
- [8] Fowler, Martin. Beck, Kent. Brant, John. Opdyke, William. Roberts, Don. (1999). *Refactoring: Improving the Design of Existing Code*. Addison Wesley. Sid 63-72.
- [9] Hunt, Andrew. Thomas, David. (1999). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional. Sid 49.
- [10] Arsenovski, Danijel. (2008). *Professional Refactoring in Visual Basic*. Hoboken, NJ, USA: Wrox. Sid 182.
- [11] *.NET Programming in Visual C++* <http://msdn.microsoft.com/en-us/library/68td296t.aspx> (hämtat: 17.3.2014)
- [12] *MFC Desktop Applications*. Microsoft Developer Network. <http://msdn.microsoft.com/en-us/library/d06h2x6e.aspx> (hämtat: 4.1.2014)
- [13] *The Java Programming Language and the Java Platform* <http://www.oracle.com/technetwork/topics/newtojava/downloads/index.html> (hämtat: 14.2.2014)
- [14] *Static Classes and Static Class Members* <http://msdn.microsoft.com/en-us/library/79b3xss3.aspx> (hämtat: 27.2.2014)
- [15] *Commanding Overview* [http://msdn.microsoft.com/en-us/library/ms752308\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/ms752308(v=vs.110).aspx) (hämtat: 17.3.2014)
- [16] *BackgroundWorker Component* [http://msdn.microsoft.com/en-us/library/vstudio/c8dce2\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/c8dce2(v=vs.100).aspx) (hämtat: 17.3.2014)

- [17] *Threading (C# and Visual Basic)*  
<http://msdn.microsoft.com/en-us/library/ms173178.aspx> (hämtat: 3.2.2014)
- [18] *Windows Communication Foundation Services and WCF Data Services in Visual Studio* [http://msdn.microsoft.com/en-us/library/vstudio/bb907578\(v=vs.100\).aspx](http://msdn.microsoft.com/en-us/library/vstudio/bb907578(v=vs.100).aspx)  
(hämtat: 24.1.2014)
- [19] *Word Object Model Overview*  
<http://msdn.microsoft.com/en-us/library/kw65a0we.aspx> (hämtat: 18.2.2014)
- [20] *Interoperability (C# Programming Guide)*  
<http://msdn.microsoft.com/en-us/library/ms173184.aspx> (hämtat: 10.2.2014)
- [21] Rybak, Martin. (2013). *Why Static Code is Bad*. Objective C#. <http://objcsharp.wordpress.com/2013/07/08/why-static-code-is-bad/> (hämtat: 17.3.2014)

# Bilagor

<b>Routine Test Report</b> V4.0						Cert. No. 123456				
Customer: ATLAS COPCO						Date of Issue 2013-04-15				
Customer ref.: 7264118 PRE-ORDER 996142						Type: M2CA 315MB 2 B5 SA Protection type: Serial no.: 0842-010773141 Tag no.: Order no.: PS-20930-1				
Rating: 3~ Motor		Product Code 3GCA311320-BXAC075								
Insul.cl. F AMB +50C IP 55 840 kg		Duty	V	Hz	kW	r/min	A	cos φ	I <sub>A</sub> / I <sub>N</sub>	t <sub>E</sub> [s]
		S1	690 D 660 D	60 60	185 185	3579 3576	184 192	0.88 0.88	1.3 0.8	0.02 0.01
Resistance U 1 - V 1 0.02635 Ω 21°C U 1 - W 1 0.02634 Ω V 1 - W 1 0.02635 Ω			Insulation resistance 25000M Ω 1000 V 21 °C			Overload test				
			High-voltage test 2400V 60 s							
Test		Line U[V] f[Hz]		Input I[A] P <sub>1</sub> [kW]		Output P <sub>2</sub> [kW] n[r/min]		Cos φ	η [%]	
No-load test		674.4 D	50	49.0	5.932	5.423	2997	0.90	0.91	
Locked-rotor test		93.2 D	50	161.9	6.402	5.701	2986	0.90	0.89	
Vibration 0.40 mm/s										
Manufactured and tested in accordance with rules of IEC 60034-1.										
On behalf of customer ATLAS COPCO										
On behalf of manufacturer 										
Date of test: 2008-11-20										
Tested by ABB Oy. Motors. Vaasa										

Computer print-out valid without signature