TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

VERKSTADSSYSTEM SOM SILVERLIGHT-APPLIKATION

Saso Kitanoski

Hannes Zügner

EXAMENSARBETE 2009

Datateknik

TEKNISKA HÖGSKOLAN

HÖGSKOLAN I JÖNKÖPING

SOFTWARE FOR ADMINISTRATION OF WORK SCHEDULES IN SILVERLIGHT

Saso Kitanoski

Hannes Zügner

Detta examensarbete är utfört vid Tekniska Högskolan i Jönköping inom ämnesområdet datateknik. Arbetet är ett led i den treåriga högskoleingenjörsutbildningen. Författarna svarar själva för framförda åsikter, slutsatser och resultat.

Handledare: Anders Carstensen

Omfattning: 15 poäng (C-nivå)

Datum:

Arkiveringsnummer:

Postadress:
Box 1026
551 11 Jönköping

Besöksadress:
Gjuterigatan 5

Telefon:
036-10 10 00 (vx)

Abstract

The exam work described in this report has been done in cooperation with System Andersson AB. The task was to investigate whether the current software for administration of work schedules, which is a windows application, would function as a web application. The web application should consist of a few functions which are clock in/clock out, work in full screen mode, create reports and function with touch screen.

The result of this exam work is a well functioning web application where all the functions is working with one exception which is the creation of reports which was not supported by the development tool.

The report describes the theoretical background for the work as well as the development of the web application and a description of how the application was tested.

Sammanfattning

Detta examensarbete är utfört i samarbete med System Andersson AB. Uppgiften var att undersöka huruvida det nuvarande verkstadssystemet, som i dagsläget är en Windowsapplikationen, skulle fungera som en webbapplikation. Webbapplikationen skulle bestå av ett antal begränsade funktioner som var instämpling/utstämpling, fungera som helskärm utan ramar från webbläsare, skapa rapporter samt fungera med pekskärm.

Resultatet blev en väl fungerande webbapplikation där alla funktioner fungerar med undantag för skapandet av rapporter vilket inte stödes av utvecklingsverktyget.

Rapporten ger den teoretiska bakgrunden och beskriver utvecklingen av webbapplikationen med bl.a. val av olika utvecklingsverktyg. Rapporten beskriver också hur webbapplikationen testats.

Nyckelord

Objektorienterat, Microsoft Silverlight, Databas, Användartest, webbapplikation, MicrosoftVisual Studio.

Innehållsförteckning

Figurförteckning

1 Inledning

Denna rapport har utförts som en del av utbildningen till ingenjör på jönköpings tekniska högskola. Arbetet gjordes tillsammans med System Andersson AB som är ett företag baserat i Jönköping. Själva arbetet passar väl in med kurserna som ingår i programmet och kompetens krävs från olika kunskapsområden för att slutföra arbetet. Kurser som är aktuella i denna rapport är bl.a. programmeringsmetoder, objektorienterad programmering, systemutveckling, database systems and applications och interaction design.
Problemet System Andersson AB har är att deras nuvarande verkstadssystem känns gammalt och System Andersson AB vill utveckla en ny version av sitt system. Själva arbetet utförs så mycket som möjligt inom ramen för systemutveckling och de teorier som finns inom det ämnet. Målet med själva rapporten är att ta reda på om det går att implementera ett verkstadssystem som är implementerat med hjälp av Microsoft Silverlight.

1.1 Bakgrund

Detta examensarbete är gjort tillsammans med System Andersson AB som är baserat i Jönköping. System Andersson AB har ett verkstadssystem idag som är utvecklat för att användas i Microsoft windowsmiljö. Företaget skulle vilja utreda om det är möjligt att använda detta system som en webblösning och om det går att använda andra miljöer än Microsoft windows, t.ex. Linux.

1.2 Företagsbakgrund

System Andersson grundades 1980 och arbetar med att utveckla ett flertal produkter under samlingsnamnet Andersson Qwick MPS. System Anderssons mål är att göra tillverkande företag effektivare. Varje dag använder över 25 000 medarbetare någon av System Anderssons produkter. Idag har System Andersson 18 anställda vilket är en fördubbling över fyra år. System Andersson är inne i en expansiv fas och har även fördubblat sin omsättning över dessa fyra år. System Andersson är marknadsledande inom verkstadssystem.

1.3 Projektsammanfattning

För att testa olika miljöer kommer en testapplikation att göras i ett tidigt skede av projektet. En testapplikation är en applikation eller program som utför olika tester och undersöker hur funktioner fungerar i en mer skyddad miljö.
Testapplikationens syfte är att ta reda på vad som fungerar och vad som inte fungerar innan man implementerar funktionerna i den slutgiltiga applikationen. Testapplikationen kommer att vara lik det verkstadssystem som System Andersson AB har idag där de anställda loggar in för att sedan välja det jobb som ska utföras.

Testapplikationen kommer ha begränsade funktioner och kommer främst att användas för att testa om det går att köra en webbapplikation som System Andersson AB har tänkt sig. Applikationen skall ha kommunikation med en databasserver samt kunna köras på en av System Andersson AB verkstadsterminal

som är utrustade med en pekskärm. Eftersom projektet omfattar en testapplikation kommer en del tid att läggas på att göra den användarvänlig då personer som inte har god datorvana kan komma att använda den slutgiltiga versionen.

System Andersson AB vill även att webbapplikationen skall kunna göra acceptabla utskrifter. Som det är nu i webbläsare så blir det i princip en skärmdump som skrivs ut på en skrivare och detta kan inte anses som acceptabelt. I detta examensarbete kommer det att utredas om det går att göra utskrifter som är annat än skärmdumpar från en webbläsare.

De stora frågorna:

- Går det att göra en webbapplikation med de krav som System Andersson AB har ställt upp?
- Kan man använda annan miljö än Microsoft Windows, t.ex. Linux för klienten?
- Fungerar en webbapplikation på en pekskärm?
- Är prestandan acceptabel?
- Går det att göra bra utskrifter?

1.4 Syfte och mål

Målet med examensarbetet är att utveckla en testapplikation för webbmiljö till Andersson Qwick MPS med begränsat antal funktioner. Dessa funktioner är:

- Applikationen ska fungera ihop med Crystal Reports.
- Pekskärm ska gå att använda till webbapplikationen.
- Ramar i t.ex. Internet Explorer ska gå att styras bort så de inte syns.

Syftet är att undersöka om en webbapplikation kan uppfylla de ovanstående målen som System Andersson AB önskar.

1.5 Avgränsningar

Testapplikation kommer att innehålla ett begränsat antal funktioner för att kunna färdigställa arbetet i tid. Den kommer att begränsas till att kunna hantera:

- Påstämpling/avstämpling med notering för tid och vilken maskin som använts
- Byta sida i webbapplikationen för att undersöka svarstider
- Visa en lista över anställningsnummer

1.6 Disposition

Denna rapport är skriven med studenter och lärare på Jönköpings Tekniska Högskola som målgrupp och kommer därför att anta att läsarna har grundläggande kunskaper i programmering och databasteorier.

Teoretisk Bakgrund

I denna del kommer teorin bakom rapporten att finnas. Vilket programmeringsspråk, databasserver, operativsystem och webbläsare som kan tänkas användas kommer att behandlas här. Användarvänlighet kommer också att tas upp som en del så testapplikationen skall vara så användarvänlig som möjligt.

Genomförande

Här kommer det att förklaras vilket programmeringsspråk som är valt, vilken databasserver, vilken systemmiljö och varför dessa valdes till förmån för andra. Avsnittet kommer även att behandla hur testapplikationen utvecklades och hur den fungerade i olika miljöer.

Resultat

Resultatet kommer att tas upp i denna del. Här kommer även en redogörelse för hur nära detta examensarbete kom de mål som var uppsatta.

Slutsats och diskussion

Problem, metodval och författarnas egna tankar kommer att finnas i denna del.

2 Teoretisk bakgrund

2.1 Teknisk Teori

2.1.1 Objektorienterad Programmering

Objektorienterad programmering baseras på objekt som man bygger upp sina program med. Traditionell programmering eller funktionsorienterad, är ett sätt att tänka på ett datorprogram som en låda som man stoppar in indata i och sedan får ut utdata. Det objektorienterade synsättet är annorlunda där man uppfattar ett program som en modell av verkligheten. Programmet är uppbyggt av objekt som är modeller av verkliga ting i applikationens omgivning. Uppgiften som utförs är då manipulering av objekten [12].

Objekt har en unik identitet och i programmet kan man komma åt dessa objekt via dess namn eller via dess fysiska minnesadress. Ett enkelt sätt att beskriva det objektorienterade synsättet är att beskriva en hiss. En hiss kan beskrivas som ett objekt. Objekt har i sin tur attribut och operationer som kan manipuleras. Dessa attribut är oftast unika för varje objekt och de brukar gömmas, eller inkapslas så att utomstående inte kan ändra dessa hursomhelst [12].

Attributen beskriver objektets tillstånd och om man använder hissen som ett exempel så kan hissen ha två attribut som anger vilken riktning hissen färdas i och vilken våning den befinner sig på. T.ex. så kan våning vara heltal som anger våningen och riktning kan ha värden som ”uppåt”, ”neråt” eller ”vilande”. Vad attribut kallas beror på språket som används, i C++ brukar attributen kallas för datamedlem [12].

När ett objektorienterat program skall utvecklas så försöker man att skriva det så effektivt och ändringsbart som möjligt. Detta görs med hjälp av att man låter modulerna i programmet bli objekt. För att ett programspråk skall få kallas objektorienterat ställer man följande krav:

- *Inkapsling* – Man skall kunna gömma detaljer som tillhör ett visst objekt så andra objekt inte kan komma åt dessa detaljer
- *Arv* – Egenskaper skall kunna ärvas från andra objekt med en arvsmekanism. Egenskaperna kan sedan anpassas för aktuella behov.
- *Generiska Programenheter* – Detta är en slags generell mall som utvecklaren kan använda för att utgå ifrån för att skapa programkod. Mallen kan anpassas för egna behov och är främst användbara när man skall skapa s.k. behållare. En behållare är ett objekt som kan innehålla andra objekt [12].

Exempel på språk som har dessa tre delar är C++ och Ada.

2.1.2 Microsoft Silverlight

Microsoft Silverlight är en del av Microsofts .NET Framework för att skapa interaktiva applikationer för webben. Silverlight är plattformsoberoende och oberoende av webbläsare. För att användare skall kunna använda Silverlight krävs ett plugin som kan laddas ner och installeras om inte användaren redan har det installerat på sin dator till webbläsaren. Versionen som används till Linux-milöer har namnet Moonlight och finns tillgängligt för nerladdning. Stöd för video och ljud finns och applikationer som utnyttjar detta kan användas. Grafik och animationer kan också användas med Microsoft Silverlight

För att skapa Silverlight-baserade applikationer kan vilket .NET Framework-språk som helst användas. Bl.a. kan C# (C sharp) användas som programmeringsspråk. Även javascript kan användas.

Silverlight har stöd för fullskärmsläge. När applikationer körs i fullskärmsläge stänger Silverlight av större delen av tangentbordet och låter endast användaren använda följande tangenter:

- Pil upp
- Pil ner
- Pil vänster
- Pil höger
- Mellanslag
- Tab
- Page up
- Page down
- Home
- End
- Enter

Detta är gjort för att minimera risken för att undvika att användaren matar in information som inte var menad att matas in. Om man har flera silverlight-plugin på samma webbsida så kan bara ett plugin köras i fullskärm åt gången. När användaren väljer att köra det i fullskärm kommer denne att få ett meddelande att trycka på *esc* för att gå tillbaka till normalläget.

Utvecklaren kan bestämma var användaren kan välja att gå till fullskärmsläge. Detta kan implementeras i princip var som helst i programkoden men dock inte i *startup event handler*. Detta är gjort så av Microsoft för att användaren alltid skall ha kontrollen över applikationen och för att förhindra skadlig mjukvara från att köras [1].

2.1.3 Microsoft .NET Framework

Enligt Microsoft är .NET ett webb-baserat verktyg för att knyta samman olika system med varandra med hjälp av mjukvara. Tanken är att .NET-baserade program skall kunna kommunicera med varandra oberoende av var de körs [13].

Microsoft .NET Framework är ett ramverk som är integrerat i Microsoft Windows och används till att bygga och köra applikationer. Ramverket erbjuder programmerare en objektorienterad miljö som kan exekvera objekt som är sparade lokalt eller via internet. Microsoft försöker även med ramverket erbjuda förenklad installation av mjukvara samt minimera versionskonflikter.

.Net Framework har två huvudsakliga delar: *Common Language Runtime* och ett .Net Framework klassbibliotek kallat *Framework Class Libraries*.

Common Language Runtime (CLR) hanterar grundläggande funktioner så som minneshantering och kommunikation. Detta innebär att programmeraren slipper koncentrera sig på saker som minnesläckage eller referenser till objekt då Common Language Runtime släpper dessa när de inte används längre [2].

Framework Class Libraries (FCL). Klassbiblioteket som är den andra stora delen av .NET Framework är en objektorienterad samling av återanvändbara objekt som kan användas för att utveckla applikationer. Dessa kan vara baserade på ett användargränssnitt som är en kommandorad eller grafiskt, så kallat Graphical User Interface (GUI). En programmerare kan t.ex. skapa egna klasser som kan användas med befintliga klasser från biblioteket [2].

.NET kan köras på olika plattformar och kan anses vara plattformsoberoende. Det kan i princip köras på vilket operativsystem som helst och har via fristående projekt tagits fram versioner för operativsystem så som Linux och Mac OS X [13].

Program som utvecklas i ramverket .NET kan konstrueras i olika programspråk där C# är ett av språken. För att utveckla applikationer inom ramverket .NET behöver man installera *.NET Framework Software Development Kit* (SDK) [13].

2.1.4 Language-Integrated Query

Language-Integrated Query (LINQ) är en del av .NET och är skapat av Microsoft som ett hjälpmedel för att användas i samband med sökning av information. LINQ utvecklades för att underlätta åtkomst till informationskällor som inte är objektorienterade som t.ex. relationsdatabaser eller XML [8].

LINQ använder istället för SQL-frågor egen syntax som är inbyggd i programmeringsspråket och kan därför användas med Visual Studio som kontrollerar syntaxen under utveckling. Detta ska underlätta utveckling av ny programvara [8].

LINQ kan inte bara användas för att hämta information från relationsdatabaser eller andra externa informationskällor utan även från vektorer.

Enkelt exempel på sökning i en vektor [7]:

```
public void Linq1() {
    int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };

    var lowNums =
        from n in numbers
        where n < 5
        select n;

    Console.WriteLine("Numbers < 5:");
    foreach (var x in lowNums) {
        Console.WriteLine(x);
    }
}
```

Utskriften på skärmen med ovanstående exempel kommer bli:

Numbers < 5:
4
1
3
2
0

Syntaxskillnaden mellan LINQ och SQL är inte speciellt stor. Hade ovanstående fråga ställts med SQL-standard så hade syntaxen sett ut på följande vis:

```
SELECT n FROM numbers WHERE n<5
```

2.1.5 Microsoft C#

Microsoft C# är ett programmeringsspråk som är designat för att göra applikationer som körs i .NET Framework och är utvecklat av Microsoft. C# skall vara enkelt, kraftfullt och objektorienterat. Microsoft C# erbjuder programmerare ett programmeringsspråk som skall vara snabbt att utveckla applikationer i samtidigt som det är baserat till stor del på programmeringsspråken C och C++ [4]. C# har även stora likheter med Java och det som kan göras med hjälp av .NET kan göras med olika Java-tekniker [3, 13].

C# är ett plattformsoberoende språk eftersom applikationer som utvecklas i språket görs inom ramverket .NET. Detta ger flera fördelar eftersom program kan utvecklas utan att programmeraren känner till hur t.ex. grafiska hjälpfunktioner fungerar i X till Linux. Detta ger möjligheten att utveckla program i Windows-miljö som sedan kan köras på andra plattformar utan att programmet behöver skrivas om för att fungera på olika operativsystem [13].

C# är objektorienterat och bygger helt på de objektorienterade principerna för att utveckla program. Applikationer skrivna i C# består då av olika objekt som samverkar med varandra. C# stöder även multitrådar som innebär att flera processer kan köras samtidigt, t.ex. kan en rörlig bild visas för användaren samtidigt som denna kan skriva in data till applikationen. C# kan också användas till att göra webbaserade program [13].

Microsoft Visual C# är ett verktyg för programmerare som vill utveckla program i C#. Verktyget erbjuder kompilator, redigerare, projektmallar, designverktyg och andra verktyg. Med hjälp av klassbiblioteket som finns i .NET Framework och som kan användas med C# får programmeraren tillgång till tjänster som finns i operativsystemet och andra användbara klasser som snabbar upp utvecklingscykeln avsevärt [4] . Detta verktyg är skapat för Microsoft Windows. Vill man utveckla på en annan plattform än Microsoft Windows finns Mono-projektet som kan laddas ner gratis och innehåller .NET Framework samt en kompilator [13].

Exempel som visar hur “Hello World” skrivs i C#:

```
using System;

namespace HelloWorld
{
  class Hello
  {
    public static void Main()
    {
      Console.WriteLine("Hello, World!");
    }
  }
}
```

2.1.6 Mono project

Mono Project är ett projekt för att skapa en UNIX-version av *Microsoft .NET Framework.* Projektet är sponsrat av Novell och är av typen *Open Source* som betyder ”öppen källkod”. Syftet är att skapa en utvecklingsmiljö för att skapa applikationer samt att köra dessa på UNIX-baserade system [6].

I Mono så ingår följande

- *Common Language Infrastructure* (CLI) som bl.a. är en virtuell maskin och *garbage collector*
- *Class Library* som innehåller både .NET klassbibliotek samt andra klassbibiliotek som Mono har lagt till.
- Kompilator för språket C#.

Idag är Mono någonstans mellan .NET 2.0 och .NET 3.5 [6].
Mono Project finns att ladda ner på http://www.mono-project.com

2.1.7 Microsoft Visual Basic

Microsoft Visual Basic utvecklades av Microsoft för att göra det lättare för utvecklare att skapa nya och användarvänliga program jämfört med språk som C eller C++. C och C++ kräver ofta flera hundra rader kod bara för att visa ett fönster på skärmen. Visual Basic kan göra samma sak fast med färre rader kod och är därför mindre tidskrävande [10].

Visual Basic ger utvecklaren möjlighet att utveckla ett program med hjälp av *drag and drop,* dvs. att utvecklaren drar ut objekt till ett fönster i utvecklingsmiljön. Därefter kan olika *events* (händelser) kopplas till dessa objekt [10].

Senaste versionen av Microsoft Visual Basic stödjer *.NET Framework* och kan användas till att utveckla applikationer inom detta ramverket. Visual Basic är också objektorienterat och kan även användas till att göra applikationer till windows, webben samt mobila enheter [5].

Exempel på hur “Hello World”-programmet ser ut i Visual Basic:

```
Module Hello World
    Sub Main()
        System.Console.WriteLine("Hello, World!")
    End Sub
End Module
```

2.1.8 Databaser

Databaser är idag en stor del av vardagen, oftast är man inte medveten om att den existerar eller att man använder den.

- Kortköp – vi användandet av kreditkort måste assistenten undersöka om det finns pengar kvar på kortet. Oftast sker detta via en kortcentral som kortet dras igenom. Centralen är i sin tur kopplad till ett datorsystem innehållande en databas med historik över senaste kortköp. Ett applikationsprogram undersöker ditt kortnummer och jämför återstående pengar med det aktuella köpet. Om köpet går igenom sparas den nya befintliga summan i databasen.
- Bibliotek – ett bibliotek har en databas innehållande böcker, utlånade böcker, reservationer, information om vem som lånat boken, etc. Ett datorsystem låter användaren söka igenom databasen efter böcker som finns på biblioteket. Systemet hanterar reservationer och tillåter användare att låna böcker.
- Resebyrå – om man ska boka en resa hos en resebyrå kommer personalen antagligen att använda och arbeta i flera olika databaser. En databas innehållande flyg och olika flygtider och en annan innehållande hotel och hotelrum. Systemet måste också hålla reda på så att personalen inte bokar in för många resande på samma flygplan eller för många personer i ett rum på hotellet. Alla dessa detaljer sparas och tas om hand i databasen [9].

Relationsdatabaser

Man kan välja att använda lite olika typer av databaser. Vanligast är att man använder en s.k. relationsdatabas där man bygger upp sin databas i form av tabeller som man sedan ger unika namn. I en relationsdatabas kan man skapa relationer mellan tabellerna vilket gör det enklare att hitta information och specifika poster, samt att bygga stora och avancerade databaser. I en relationsdatabas innehåller varje post i en tabell en rad i tabellen, dvs en uppsättning relaterade värden [14].

Objektorienterade databaser

En relationsdatabas är bra då man ska hantera mycket data som har en enkel struktur som t.ex. text eller siffror. Men om man använder mer komplicerade data från t.ex. ett CAD-system är det bättre med en objektorienterad databas.

En objektorienterad databas är som ett objektorienterat programmeringsspråk fast med en implementerad databasfunktion. I en objektorienterad databas handlar det om hur objekten motsvarar saker i verkligheten och där de olika typerna av saker beskrivs som klasser.

I en relationsdatabas kan man endast hitta och söka efter data i en tabell baserat på vad just en rad i tabellen innehåller. Skillnaden i en objektorienterad databas är att varje objekt som sparas i databasen tilldelas ett unikt objekt-id som det senare går att söka efter. Ett objekt i en objektorienterad databas kan innehålla information om många saker som t.ex. information om andra objekt, scheman, listor, etc. till skillnad från en relationsdatabas där varje rad i en tabell endast innehåller en textsträng, numeriskt värde eller liknande [14].

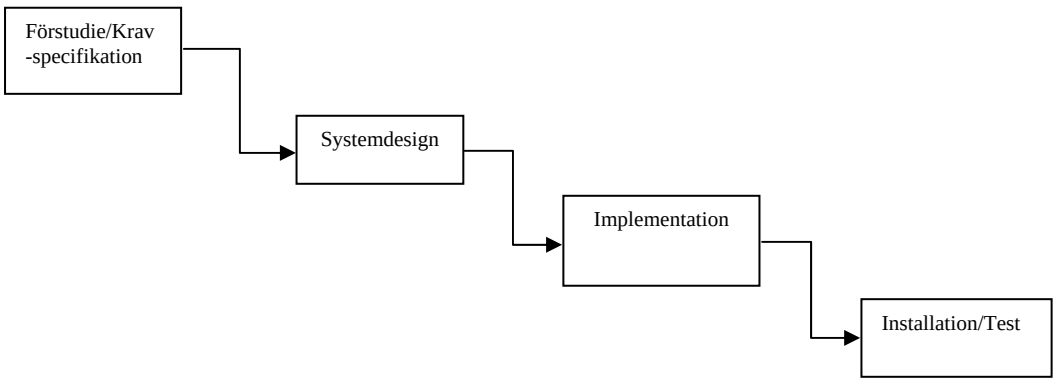
Exempel på relationsdatabaser och objektorienterade databaser är,
MySQL som är en av de största *open source* databaserna, SQL Server 2008 som är en produkt från Microsoft, Oracle Database som är en relationsdatabas från Oracle Corporation och ObjectStore som är en objectorienterad databas från Progress.

2.1.9 Systemutveckling

Inom systemutveckling finns två huvudmetoder. Den ena metoden är den s.k. ”Vattenfallsmetoden” och den andra metoden kallas ”iterativ systemutveckling” eller ”Agile systemutveckling”. Vattenfallsmetoden används inte så ofta nu för tiden då det är ett förlegat tillvägagångssätt. Vattenfallsmetoden ger ofta en mindre uppskattad slutprodukt än vad den iterativa metoden gör.

Vattenfallsmodellen

I vattenfallsmodellen följer man ett visst antal givna steg, se figur 1. Namnet har modellen fått då processen liknar ett vattenfall. Man börjar i toppen och arbetar sig nedåt. Nackdelen med denna metod är att den endast har en riktning, vilket är nedåt, när man gått förbi ett av stegen går man inte tillbaka.



Figur 1 Ett exempel på ”Vattenfallsmodellen”.

Detta schema följs oftast när man använder vattenfallsmetoden. Modellen ovan är en enkel version, det finns mer avancerade modeller av denna metod.

Problemet med denna metod är att användarna och slutkunderna ofta har väldigt lite att säga till om under utvecklingsstadiet. Alla stora och viktiga beslut tas under första fasen som är förstudie/kravspecifikation. Efter denna fasen är användarna sällan inblandade.

Vattenfallsmetoden ser ut att ge en stor trygghet då mycket tid läggs på förstudien, när det senare visar sig att vissa funktioner inte fungerar går man inte tillbaka för att rätta till detta i förstudien utan slutprodukten blir något helt annat än beställaren önskat.

Iterativ systemutveckling

Den andra kända metoden inom systemutveckling är den iterativa metoden eller ”Agile metoden”. Denna metod bygger på att man tar en liten del av processen och gör den klar för tester i verkligheten. Man sätter den delen man är klar med i drift och låter användare testa den för att sedan samla in synpunkter och åsikter från dessa. Efter testfasen börjar man om från början och ändrar de funktioner som användarna hade synpunkter om. Denna cykel håller på tills man fått ett acceptabelt resultat.

Man har även med denna metod en förstudiefas där man samlar in information och tar fram en kravspecifikation. Denna fas blir dock inte lika omfattande från början utan man kan gå tillbaka och göra om ifall man inte blir nöjd direkt.

Fördelen med denna metod är att utvecklare och slutanvändare hela tiden är på samma nivå och har samma uppfattning om hur slutprodukten kommer att se ut. Då man hela tiden tar det ett steg i taget och stämmer av med slutanvändaren genom utvärderingar [15].

2.1.10 Crystal Reports

Det går alltid att skapa data i rapportform genom att ”loopa” igenom olika poster eller protokoll och sedan skriva ut informationen i din windowsapplikation eller webbapplikation.

Men att presentera informationen i snygga rapporter är desto svårare. Crystal Reports gör det lätt att skapa komplexa och professionella rapporter i ett program med grafiskt användargränssnitt. Rapporterna går också att ansluta till nästan vilken databas som helst. Med den inkluderade guiden är det lätt att bestämma format, gruppera och ordna efter egna krav.

Med Crystal Reports kan du skapa rapporter till både windowsapplikationer och webbapplikationer. Rapporten kan sparas ner till olika format som t.ex. PDF, text eller XML [4].

2.2 Användbarhet

För att få ett fungerande och tillfredställande system behöver man veta vad slutanvändaren förväntar sig av systemet. Detta görs vanligtvis genom att man kommer fram till en gemensam kravspecifikation. Många system och projekt idag blir misslyckade på grund av dålig eller ingen användbarhet. Därför är det extra viktigt att involvera de egentliga slutanvändarna i projektet.

Utvecklingen av ett nytt system går vanligtvis igenom ett antal faser som är:

- Utformning av kravspecifikation
- Design
- Konstruktion/Implementation
- Utvärdering

Kravspecifikation

Innan man börjar ett projekt tar man fram en kravspecifikation över vad systemet ska innehålla och kunna göra. Man måste förstå så mycket som möjligt om användarna och deras arbete då målet med systemet oftast är att underlätta för just användarna.
Innan man börjar designa och skriva programkod är det därför viktigt att först ha klart för sig vad det är man ska göra. Kravspecifikationen kan variera i längd beroende på vilken typ av metod och genomförande man använder.

För att komma fram till en bra kravspecifikation måste man samla in data från användarna och det finns ett antal olika tillvägagångssätt för detta:

- Intervjuer – kan göras olika, antingen har man färdigkonstruerade frågor eller en mer öppen diskussion.
- Frågeformulär – brukar användas inledningsvis för att få en översikt om vad användarna tycker, sedan väljs ett antal för djupare intervjuer.
- Direkt observation – man studerar användare i deras naturliga miljö för att få en uppfattning om deras arbete och uppgifter.
- Fokus grupper – används vid mer komplexa problem där det finns meningsskiljaktigheter och man behöver få en djupare insikt.

Design

När man har färdigställt kravspecifikationen går man vidare till designfasen. Första steget är att göra en enkel prototyp över systemet, en som förmodligen inte kommer se ut som slutprodukten. Att göra en enkel prototyp är väldigt användbart vid diskussion med slutkunden, då kan man få feedback på om man är på rätt spår eller inte. En enkel prototyp svarar på frågor och hjälper användarna att ge feedback. En stor fördel med enkla prototyper är att de är billiga att skapa samtidigt som de ger bra feedback.

Denna enkla prototyp kan göras med olika medel.

- *Storyboard* – innehåller ett antal bilder som visar ett typiskt arbetsflöde för en användare.
- *Wizard of Oz* – ett annat sätt att ta sig an en enkel prototyp förutsätter att man har utvecklat en enkel programvara att ha till sin hjälp vid denna typ av testning. Med denna teknik sitter användaren framför en dator och testar programvaran som om det skulle vara den slutgiltiga produkten. Det användaren inte vet är att den datorn han sitter vid är ansluten till en annan dator där det sitter en operatör och simulerar svaren från programvaran.

Efter man har kommit överens om en enkel prototyp går man vidare till att skapa en avancerad prototyp som mer liknar slutprodukten. Det innebär t.ex. att man kan för den avancerade prototypen skapa en ny programvara i Visual Basic istället för som innan via t.ex. en *storyboard*. Dessa avancerade prototyper är användbara vid möten med slutkund då denna får en bättre insikt i hur programvaran kommer att se ut. Prototypen är också bra då man testar olika tekniska förutsättningar. Här är användartester mycket viktiga då man ofta får bra respons från användarna om där skulle saknas någon funktion som är viktig i deras dagliga arbete.

Användartester

Dessa tester är mycket viktiga för att få en fungerande webbapplikation och en nöjd slutkund. Då ett användartest ska utföras väljs en specifik grupp ut som ska genomföra och testa olika funktioner med webbapplikationen. Under tiden försöksgruppen utför sina uppgifter noterar utvecklare och andra observatörer hur väl produkten fungerar.

Användartester görs oftast i flera olika faser. Under den första fasen har man oftast ett antal frågor som man vill ha svar på. Dessa kan vara:

- Hittar användaren rätt från början i webbapplikationen eller behöver han leta under en längre tid innan han hittar det han söker.
- Gör användaren mer än ett fel innan han hittar det han söker.
- Hittar användaren tillbaka då han har gjort ett fel.

Under tiden måste man också försöka tolka användarens uppskattning/missnöje med produkten.

Då man rättat till diverse fel efter första fasen av tester ställer man sig nya frågor inför fas två t.ex.

- Klarar användaren av att utföra uppgiften under en viss utsatt tid.
- Visar användaren goda intentioner på att vilja använda webbapplikationen i framtiden.

Det är viktigt att börja med användartester i ett tidigt skede i utvecklingen. Ju tidigare man börjar desto fler fel kan rättas till innan slutproduktionen börjar, detta leder också till en nöjdare slutkund.

Viktigt att komma ihåg under användartest är att man testar webbapplikationen och inte användarna. Det är utvecklarnas uppgift att göra en såpass bra webbapplikation att användarna ska klara av att använda den.

Antalet användare för testerna kan variera beroende på vad det är man producerar. Är det så att en webbapplikation har många olika användare med varierande datorkunskap måste man inkludera användare från alla grupper av datorkunskap.

Konstruktion/Implementation

När man är klar med sina enkla och avancerade prototyper och man känner sig säker på att dessa uppfyller alla krav tar man med sig allt man har lärt sig och börjar skapandet av den slutgiltiga produkten [11].

3 Genomförande

3.1 Utvecklingsmiljö

För att kunna utveckla en webbapplikation krävs det att man bestämmer sig för vilken typ av mjukvara man skall använda. Mjukvara för applikations-utveckling kan skilja sig avsevärt från varandra och dessa skillnader kan vara avgörande för om ett program är lämpligt att använda.

3.1.1 Val av databas

Valet av databas var bestämt redan från början. Vid ett möte med System Andersson AB bestämdes att det var Microsoft SQL Server 2008 som skulle användas.

3.1.2 Val av programmeringsspråk

Programmeringsspråket som används är C# då det stöds av både .NET Framework och Silverlight. Eftersom applikationen skall vara gjord med hjälp av Silverlight så är valet av C# naturligt då det både känns kraftfullt och har stöd för användning av databaser.

3.1.3 Val av utvecklingsverktyg

Prototypen är utvecklad i Visual Studio Web Developer 2008 Express. I detta program har C#-koden skrivits samt grundläggande layout gjorts. För att göra användargränssnittet mer användbart har sedan Microsoft Expression Blend 2 använts.

3.1.4 Utformning av kravspecifikation

När System Andersson AB kom fram till att deras nuvarande system skulle köras via en webbapplikation behövdes en kravspecifikation utformas för webbapplikationen. Detta är en viktig del vid skapandet av ett nytt projekt eller system. Kravspecifikationen skall beskriva de krav som System Andersson AB har på webbapplikationen när den är färdig.

Kraven fastställdes efter en diskussion med medarbetare på System Andersson AB. Kravspecifikationen innehåller en projektbeskrivning, mål, avgränsningar och ett antal krav i form av funktionella krav, utskriftskrav och prestandakrav som alla listades i kravspecifikationen.

Tidigt under projektet skapades en egen webbläsare i C# för Microsoft Windows för att kunna hantera fullskärmsläge på ett tillfredställande sätt. För att göra detta till Linux krävdes ytterliggare kunskaper och tid, därför implementerades aldrig en Linux-version av den egna webbläsaren och inga tester gjordes då av klienten i Linuxmiljö.

Se kravspecifikation i bilaga 1.

3.2 Prototyp

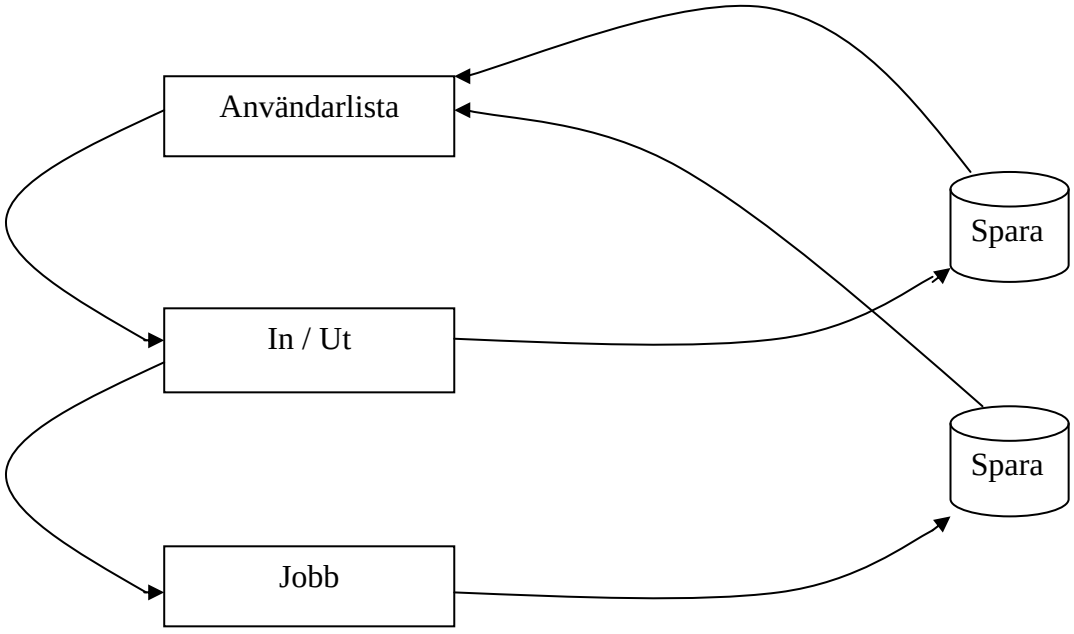
Prototypen som utvecklades var en enkel programvara vars syfte var att testa om det är möjligt att använda .NET och Silverlight till System Andersson AB verkstadssystem. En kortare förstudie gjordes samt en kravspecifikation innan arbetet med själva prototypen startade.

3.2.1 Förstudie till prototyp

Innan prototypen utvecklades gjordes en mindre testapplikation för att göra oss bekanta med utvecklingsmiljön eftersom erfarenheten av Visual Studio Web Developer 2008 Express var liten. Detta gäller även Microsoft SQL Server 2008 Express.

3.2.2 Webb-applikationen

Till grund för prototypen användes kravspecifikationen som gjordes tillsammans med System Andersson AB.



Figur 2 flödesschema för Prototypen

Enligt kravspecifikationen skulle flödet ske som i figur 2. Användaren ser en lista på användare. Användaren klickar på knappen med sitt eget namn och får upp en ny skärm där valet att stämpla in visas. Knappen stämpla in kommer bara att visas om användaren inte är instämplad och knappen stämpla ut kommer bara visas om användaren är instämplad. När valet att stämpla in görs skrivs instämplingen in i tabellen TimeIn och listan över anställda visas igen. När användaren är klar med jobbet väljer han att stämpla ut och jobbet skrivs in och sparas i tabellen TotalTime.

Själva applikationen gjordes med hjälp av verktygen Visual Studio Web Developer 2008 Express och Microsoft Expression Blend 2. All grafik gjordes med hjälp av Blend och all programkod skrevs i Web Developer. Båda programmen körs samtidigt och har samma projekt öppet. Om man ändrar något i projektet i t.ex. Web developer kommer Blend att fråga användaren om projektet ska laddas om. På detta vis kan båda programmen ha senaste version av projektet öppet samtidigt. Detta innebär att man kan jobba i båda programmen samtidigt då ändringar registreras av båda programmen.

På förstasidan möts användaren av en lista på alla användare som finns i systemet. Dessa skall vara röda eller gröna beroende på om personen är ”instämplad” eller inte. Dessa användare gjordes som knapp-objekt med en ram som kan byta färg till rött eller grönt, se figur 3 och 7.

För att göra detta enkelt och så nära verkligenheten som möjligt gjordes en klass Emp, som skall vara en förkortning av engelska ordet Employee (se bilaga 3). Denna klass är väldigt lik tabellen Employees i databasen. När programmet startas läses alla användare in från databasen och sparas i en *array* av typen emp. Alla objekt av typen emp håller reda på om personen är ”instämplad”. För att hämta alla användare från tabellen Employees används följande fråga med hjälp av *LINQ:*

```
public List<Employee> GetEmployeeByEmployeeID(int eId)
{
            //Gets employees from employee by epmloyeeid

            DataClasses1DataContext db = new
            DataClasses1DataContext();
            var matchingEmployees = from empList in db.Employees
                                    where empList.EmployeeID < eId
                                    select empList;
             return matchingEmployees.ToList();

}
```

Anledningen till att mindre än används är för att metoden skrevs först och behovet av att få ut alla användare upptäcktes först senare. Istället för att skriva en helt ny metod för att välja ut alla användare ändrades:

where empList.EmployeeID = eId

till:

where empList.EmployeeID < eId

När knapparna visas för användaren kommer en kontroll att göras för att visa rätt färg på ramen.

```
//show correct color on border
if (emp[0].getStamp())
{
            btnEmployee1.BorderBrush = new
            SolidColorBrush(Color.FromArgb(255, 7, 139, 0));
}
else
{
            btnEmployee1.BorderBrush = new
            SolidColorBrush(Color.FromArgb(255, 168, 10, 10));

}
```

btnEmployee1 är själva knappen och metoden BorderBrush ändrar färgen på själva ramen runt knappen. En if-sats kontrollerar om personen är ”instämplad” med hjälp av metoden getStamp(), se figur 3.

Figur 3 Användare med grön och röd ram

Längst uppe till vänster i programmet skall det finnas datum och aktuell tid. I C# kan man använda en *Timer* för att räkna åt en men i Silverlight finns inte denna *control*. För att lösa detta skapas en metod som visar tiden på skärmen genom att skriva ut en sträng med dagens datum och klockslag. Denna metod anropas av en annan metod som kallas StartTimer varje sekund. Resultatet blir att dagens datum och tid uppdateras varje sekund på skärmen. Nedan följer koden i metoden StartTimer som anropas när programmet startas.

```
System.Windows.Threading.DispatcherTimer myTimer =
new System.Windows.Threading.DispatcherTimer();
myTimer.Interval = new TimeSpan(0, 0, 0, 1); // 1 second
myTimer.Tick += new EventHandler(oneSecTick);

myTimer.Start();
```

Varje sekund som programmet körs kommer metoden oneSecTick att anropas där datum och tid skrivs ut på skärmen.

När en användare väljer att stämpla in kommer följande kodrad att exekveras för att skriva in data till databasen i tabellen TimeIn. För en överblick på hur databasens design ser ut se avsnitt 3.2.4.

```
public void insertIntoTimeIn(int empId, int tId, int orderNumber)
        {
            //Inserts time into TimeIn

            DataClasses1DataContext db = new
            DataClasses1DataContext();
            DateTime current = DateTime.Now;

            // Create a new row object.
            TimeIn row = new TimeIn
            {
                EmployeeID = empId,
                TaskID = tId,
                ClockIn = current.TimeOfDay,
                DateIn = current.Date,
                OrderId = orderNumber
            };
            // Add the new object to the collection.
            db.TimeIns.InsertOnSubmit(row);

            // Submit the change to the database.
            db.SubmitChanges();

    }
```

När användaren väljer att ”stämpla ut” kommer denna rad från tabellen att tas bort och skriva in en ny rad i tabellen TotalTime. Tabellen TimeIn fungerar då i praktiken som mellanlagring medan användaren arbetar.

All grafik som finns i applikationen är egentillverkad. Detta gjordes enkelt och smidigt med hjälp av Microsoft Expression Blend 2. Knapparna gjordes som templates vilket förenklade utvecklingen avsevärt då det räckte med att göra en knapp av varje sort och göra rätt inställningar så som när knappen är nertryckt, avstängd osv. När detta var gjort för en knapp kunde detta lätt kopieras till andra knappar och på så vis sparades massvis av tid.

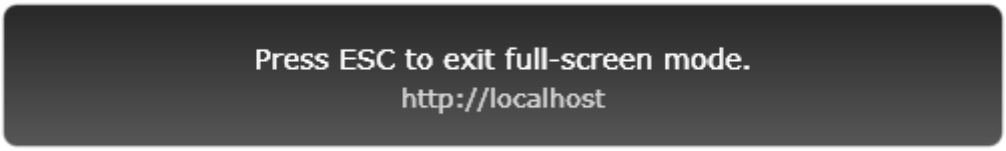
3.2.3 Alternativa webb-läsare för applikationen

System Andersson AB ville att klienten skulle köras i en webb-läsare och att sökfältet skulle vara borttaget för användaren eller att klienten skulle köras i fullskärmsläge. Olika alternativ på hur detta skulle lösas togs fram och är

- Applikation i fullskärmsläge
- Internet Explorer utan menyer och sökfönster
- Egenutvecklad webb-läsare

Applikation i fullskärmsläge

Ett av alternativen var att köra applikationen i fullskärmsläge då detta stöds av Microsoft Silverlight. Problem uppstod dock när stöd för fullskärm inte ges via programkod. För att kunna köra applikationen i fullskärmsläge måste användaren själv ha valt detta genom en aktiv handling, t.ex. klickat på en knapp som säger till att applikationen skall köras i fullskärm. Vid en knapptryckning på en sådan knapp kommer applikationen växla till fullskärm och användaren kommer bli uppmärksammad på detta genom ett meddelande på skärmen (se figur 4).



Figur 4. Meddelande till användaren.

För att få en applikation att visas i fullskärm så används följande kodrad för att växla mellan fullskärm och vanligt läge då användaren gör sin aktiva handling:

```
Application.Current.Host.Content.IsFullScreen =
!Application.Current.Host.Content.IsFullScreen;
```

Denna rad kod gör så att isFullScreen får det negerade värdet av sig själv. Om isFullScreen, som är en boolesk variabel, är satt till sann kommer den att ändras till falsk och tvärtom. På detta vis kan man ha en enda knapp som ger användaren möjlighet att växla mellan fullskärm och normalt läge och kodmängden blir så liten som en rad.

Eftersom användaren själv aktivt måste välja fullskärmsläge genom en knapptryckning så räcker det inte att lägga in ovanstående kodrad i metoden som körs när man startar applikationen. Detta är stoppat av säkerhetsskäl av Microsoft. Om man skriver in raden ändå så kommer den helt enkelt att ignoreras när programmet körs.

En möjlighet att gå runt detta problem är att skapa en välkomstsida som välkomnar användaren till systemet och att denne måste klicka på någon form av ”startknapp” för att ”starta” programmet som växlar över till fullskärmsläge. Detta kan göras på flera sätt. Ett av förslagen är att skapa en liten och enkel applikation i C# som endast består av ett fönster och en *webcontrol* som vid uppstart av programmet alltid visar webb-sidan med silverlight-applikationen. I denna lilla ”startapplikation” kan användaren välkomnas till programmet och uppmanas att klicka på start som då växlar över till fullskärmsläge. Meddelandet om att man ska trycka på ”ESC” för att växla tillbaka till fönsterläge kommer dock vara kvar. Ett problem är dock att inga inmatningar kan göras när det körs i fullskärmsläge på detta vis. Microsoft har byggt in ett skydd så att bara ett fåtal tangenter kan

användas (se avsnitt om Microsoft Silverlight). Denna begränsning gör att detta alternativ inte är så lämpligt då inmatningar från tangentbord i princip är ett måste.

En annan nackdel med denna metod är att användaren blir tvingad till två steg innan applikationen startar, först måste man klicka på ikonen och när startapplikationen visas åter igen klicka på en knapp för att starta.

Internet Explorer utan menyer och sökfält

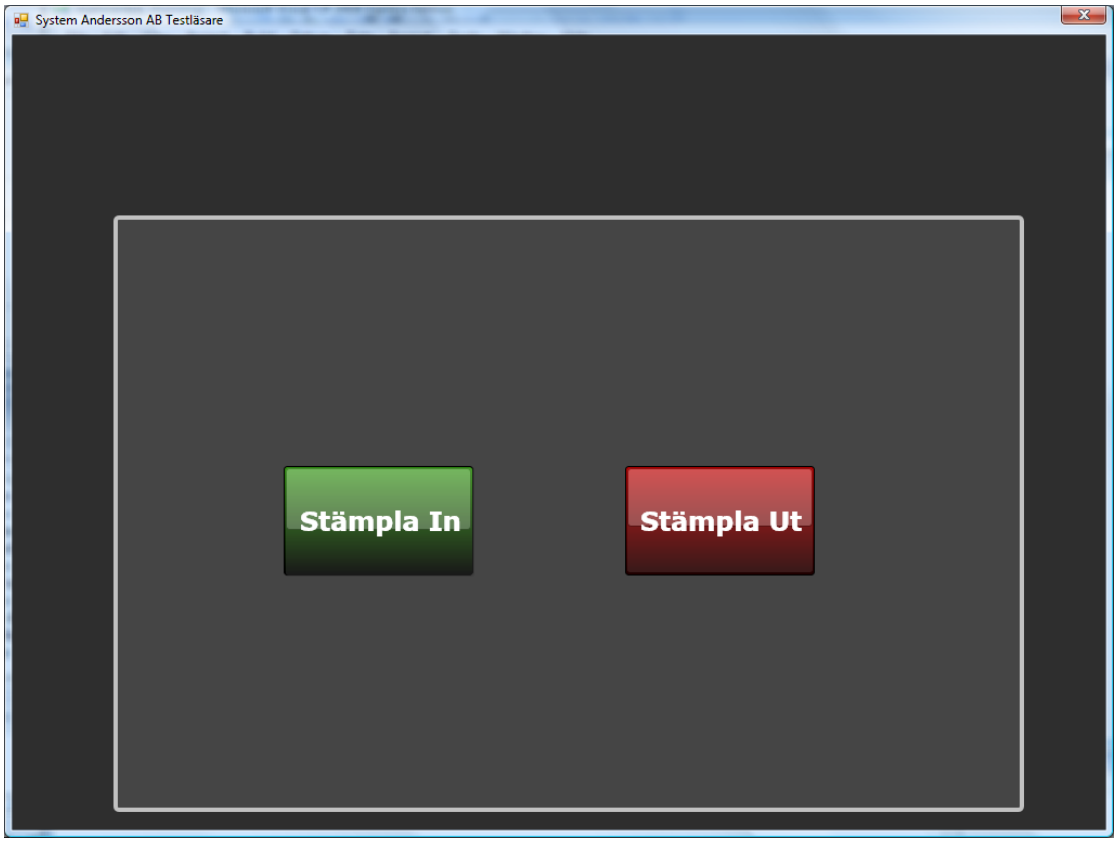
Ett annat alternativ som diskuterades med System Andersson AB gick ut på att skala av Microsoft Internet Explorer så att bara ramen fanns kvar och hindra användaren från att se sökfält och menyrader. Detta är dock inte möjligt fullt ut. Det går att skala bort statusfältet och menyraden men inte sökfältet.

Detta går dock att komma runt genom att man utvecklar en egen enkel webb-läsare i Visual Studio C# 2008.

Egenutvecklad webb-läsare

För att gå runt problemet med att Internet Explorer inte är fullt konfigurerbart så kom idén att utveckla en egen enkel webb-läsare i Visual Studio C# 2008.

Tanken är att man skapar ett fönster med en *webcontrol* som i grunden är Internet Explorer. När användaren klickar på ikonen för att starta systemet på skrivbordet så startas den enkla webbläsaren och visar programmet med enbart en ram runt *webcontrol* (se figur 5).



Figur 5. Exempel på en egen webbläsare som kör en enkel silverlight-applikation

Fördelen med denna metod är att man kan forma fönstret efter egna behov. Inmatningar från tangentbord fungerar med denna lösning och meddelandet att programmet körs i fullskärm behövs inte då fönstret kan anpassas till att täcka hela skärmen när det startas. En annan fördel är att man kan stänga av funktioner som minimera och maximera i fönstret så att användaren inte kan göra något annat än att stänga ner programmet.

System Andersson AB vill köra programmet i fullskärm och med hjälp att stänga av alla ramar på fönstret och sedan maximera det kommer det att köras i fullskärm. För att åstadkomma detta kan man lägga till följande rader kod som ger fullskärm och att applikationen hamnar överst av alla fönster.

```
//set to fullscreen and on top
```

```
this.Bounds = Screen.PrimaryScreen.Bounds;
this.TopMost = true;
```

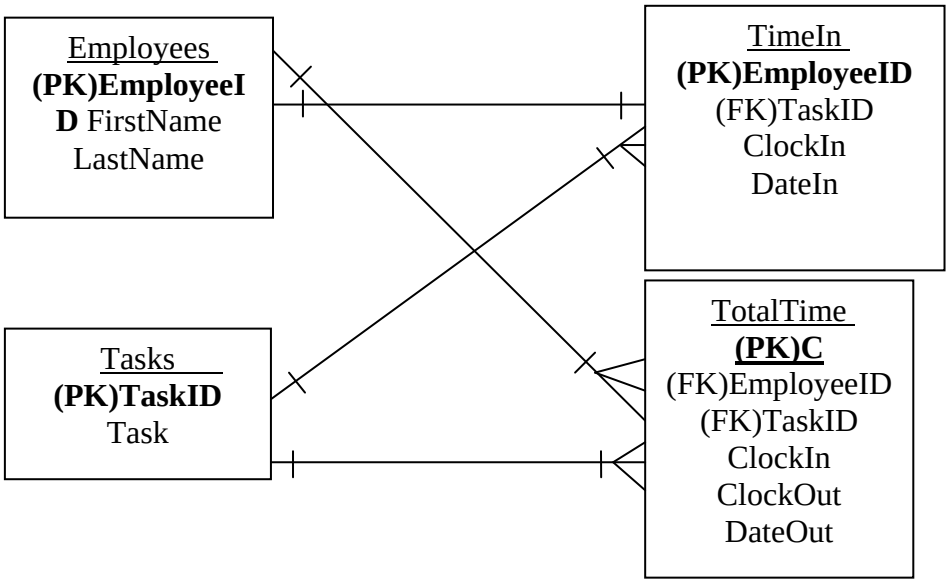
3.2.4 Produktion av databas

Från början skapades tabellerna utifrån ett *Graphical User Interface* (GUI) vilket senare visade sig vara en dålig idé då det senare var svårt att skapa relationer mellan tabellerna. Det visade sig att det var mycket enklare att skapa tabellerna med hjälp av kod. Efter ett par dagar var kunskapen såpass god att en enkel databas med de nödvändiga tabellerna och relationerna kunde skapas. De två SQL-frågorna nedan beskriver hur skapandet av tabeller och relationer fungerar i Microsoft SQL Server 2008.

```
CREATE TABLE dbo.Employees(EmployeeID INT PRIMARY KEY, FirstName
Char(10), LastName Char(10));

CREATE TABLE dbo.TimeIn(EmployeeID INT PRIMARY KEY REFERENCES
dbo.Employees(EmployeeID)
, TaskID INT FOREIGN KEY REFERENCES dbo.Tasks(TaskID), PunshedIn
time(7), DateIn date);
```

Applikationen som producerades hade två huvudfunktioner som var ”stämpla in” och ”stämpla ut” så databasen behövde totalt fyra enkla tabeller. En tabell över anställda som heter ”Employees” där de anställda listas med unikt anställningsnummer samt för- och efternamn, en tabell över tid för instämpling som döptes till ”TimeIn” där det är tänkt att en anställd endast kan vara instämplad med ett jobb åt gången och sedan plockas bort ur den tabellen vid utstämpling och flyttas till ”TotalTime” som sparar totala tiden efter utstämpling och sist en tabell över diverse jobb som kan tänkas utföras som döptes till ”Tasks” där jobben sparas med unikt ”TaskID” samt vilket jobb det är. För att se hur databasens tabeller och relationer av konstruerade se figur 6.



Figur 6 överblick på hur databasen tabeller och relationer av konstruerade.

Då applikationen skulle använda databasen i ett nätverk blev det en del bekymmer med att få kontakt med databasen.
Först steget var att skapa en användare till applikationen med tillåtelse att logga in. Denna användare behövde sen tilldelas rättigheter för att modifiera databasen. Då SQL Server var inställd på att endast tillåta inloggning via *Windows Authentication* (alltså endast den person som var inlogagd på datorn fick tillgång till databasen) behövdes det ändras så att även utomstående användare kunde logga in via *SQL Authentication.* Detta hjälpte dock ändå inte för att få kontakt med databasen över nätverket utan det problem som kvarstod var att SQL Server 2008 inte tillåter anslutning via TCP/IP som *default* vilket SQL Server 2005 gör. När detta var inställt korrekt så fungerade databaskopplingen till applikationen.

3.2.5 Koppling mellan mjukvara och databas

En applikation utvecklad för Silverlight måste använda LINQ och *web service* för att kunna kommunicera med en databas. En längre förstudie fick göras då det upptäcktes att det var tvunget att göras på detta vis. Från början var det tänkt att en klass skulle skapas med metoder för att hämta data från databasen och skriva data till databasen. Då silverlight bara stöder LINQ fick den ursprungliga idén läggas ner. Lösningen med LINQ påminner väldigt mycket om den ursprungliga tanken med en klass som skulle sköta kopplingen mellan mjukvaran och databasen så själva designen på hur programmet är uppbyggt påverkades inte speciellt mycket.

3.2.6 Implementering av Crystal Reports

Då Crystal Reports sades fungera ihop med Microsoft Silverlight gjordes ett försök till att implementera detta i applikationen. Efter hand så upptäcktes att *Crystal Report Viewer* och liknande komponenter inte dök upp i ”toolboxen” i Microsoft Silverlight. Detta berodde på att Microsoft Silverlight inte stöder Crystal Reports.

3.3 Användartester

Eftersom applikationen fick ett nytt utseende och en ny design planerades ett antal användartester in för att undersöka hur väl användarna klarade av den nya designen. Målet var från början att få tag i ett företag som i dagsläget använder verkstadssystem från System Andersson AB men inget företag kunde ställa upp på ett användartest. Istället valdes ett antal testpersoner med varierande datorvana ut från Tekniska Högskolan i Jönköping samt Internationella Handelshögskolan i Jönköping. Ett testprotokoll utformades som de tänkta användarna skulle utföra under en viss tid. Ett protokoll för utvecklarna togs också fram med stödfrågor om hur väl användarna lyckades med uppgifterna

med utrymme för eventuella kommentarer om användarnas uppskattning/missnöje, se protokollen i bilaga 5.

Användarna fick inga instruktioner om hur programmet fungerar eller används innan testet började, alla var helt ovetande om vad för program det handlade om och vad som skulle göras. Användarna tilldelades testprotokollet med ett antal uppgifter som de skulle lösa. Uppgifterna var tänkbara scenarion för företaget. Under tiden studerade utvecklarna användarnas framfart i programmet och noterade tid samt eventuella fel användaren gjorde.

Efter testet frågades användarna hur det upplevde systemet och om det fanns synpunkter på något som kunde förbättras och bli mer användarvänligt.

4 Resultat

I detta avsnitt kommer resultatet av projektet att redovisas. Avstämning mot kravspecifikation görs samt redovisning av den specialanpassade webb-läsaren, databasen och användartester.

4.1 Avstämning mot examensarbetets mål

Vid avstämning mot examensarbetets mål kan sägas att de flesta av målen har uppnåtts. Ett mål som inte uppnåtts var att testapplikationen skulle fungera ihop med Crystal Reports.
Målet med att ramar skulle styras bort från webbläsaren har uppnåtts och fungerar bra. Samtliga funktioner som testapplikationen skulle ha fungerar bra, påstämpling/avstämpling med avseende på tid och datum fungerar och registreras korrekt i databasen. Sidbyten i testapplikationen fungerar utan några långa svarstider. Listan över anställda presenteras korrekt vid uppstart av testapplikationen. Applikationen visades upp för handledare och personal på System Andersson AB och dessa var nöjda med resultatet.

4.2 Prototyp avstämd mot kravspecifikation

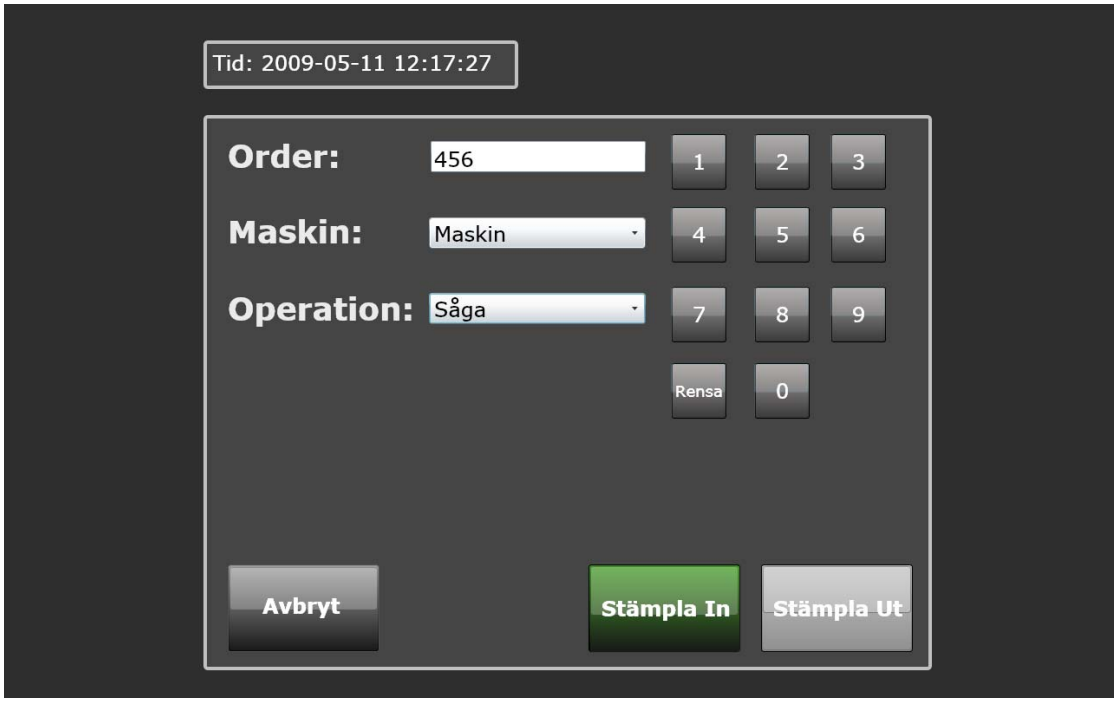
Prototypen som gjordes för System Andersson AB fungerar enligt de krav som ställdes(se kravspecifikation i bilaga 1):

Stämpla in

För att kunna välja jobb måste användaren stämpla in.

Fungerar fullt ut. Användaren kan välja sig själv på skärmen och sedan välja jobb och ange ordernummer (se figur 7 och figur 8).

Figur 7. Skärmen som användaren ser som förstasida.



Figur 8. När en användare klickat på sin egen knapp visas ovanstående skärmbild.

När en användare är ”instämplad” kommer ramen runt knappen hos den aktuella användaren att vara grön istället för röd (se figur 9).

Figur 9. Skärmen som användaren ser som förstasida när det finns instämplade användare.

När en användare är klar med sitt jobb och är redo att ”stämpla ut” flyttas data från TimeIn och kompletteras med nödvändig information och skrivs sedan in i tabellen TotalTime (se figur 10).

Figur 10. Skärmen som användaren ser när valet att stämpla ut är alternativet.

Tid/-maskinregistrering
Tid för ”instämpling” ska registreras. Val av maskin ska även det kunna registreras.

Fungerar fullt ut. Användaren som klickar på ”Stämpla in” skrivs in i tabellen TimeIn där aktuell maskin, användar-id, ordernummer och tid sparas.

Bläddra sida
Det skall finnas möjlighet att bläddra en sida för att undersöka hur detta fungerar.

Programmet byter aldrig sida och det finns ingen anledning till att byta sida eller bläddra i webb-läsaren. Webb-läsaren som gjordes i Visual Studio C# 2008 Express för just detta projekt ger inte heller användaren någon möjlighet att navigera sig bort eller att förstå att det är en webb-applikation som körs.

Crystal report
Crystal Reports stöds inte av Microsoft Silverlight. Alternativ till Crystal Reports fanns att läsa om på Silverlight.net forum. Ett alternativ utvecklas av Perpetuum Software LLC och heter Report Sharp-Shooter.

Pekskärm
Webbapplikationen ska kunna hanteras med pekskärm.

Systemet är inte testat på pekskärm men användargränssnittet är utvecklat på ett sådant vis att det skall vara möjligt att använda fingrarna för att navigera med. Storlek på knappar är gjorda tillräckligt stora för att vara användbara. Programmet kördes på en dator utan pekskärm med skärmupplösningen 1280*800 och System Andersson AB kommer att köra programmet på ännu större skärmar.

Webbläsarens ramar
Ramarna till webbläsaren ska kunna styras bort.

Fungerar fullt ut. En enkel webb-läsare gjordes i Microsoft Visual Studio C# 2008 Express där fönstrets ramar togs bort och själva fönstret ställdes in så att det täcker hela skärmen och ger användaren en känsla av att programmet körs i fullskärmsläge.

Streckkodsläsare
Systemet ska kunna klara av att använda streckkodsläsare.

Inte testat. Ingen möjlighet att testa med streckkodsläsare fanns. Dock går det att använda tangentbordet som inmatning även om programmet körs i fullskärmsläge. Detta fungerar eftersom en egen webb-läsare skapades istället för att köra själva silverlight-applikationen i fullskärmsläge.

Sammanfattning

Själva programmet uppfyller de krav som System Andersson AB ställde. Det går att ”stämpla in” och ”stämpla ut”. Tabellen TimeIn mellanlagrar information medan en användare arbetar och när användaren är klar flyttas informationen till tabellen TotalTime. Raden som fanns i TimeIn tas bort. Programmet körs även över hela skärmen och detta tack vare att webb-läsaren är specialgjord för detta projekt.

4.3 Webb-läsare

Som lösning på vilken webb-läsare som skulle användas valdes en egen version av Internet Explorer. Detta löstes med hjälp av en egen liten webb-läsare skapad i Microsoft Visual Studio C# 2008 Express.

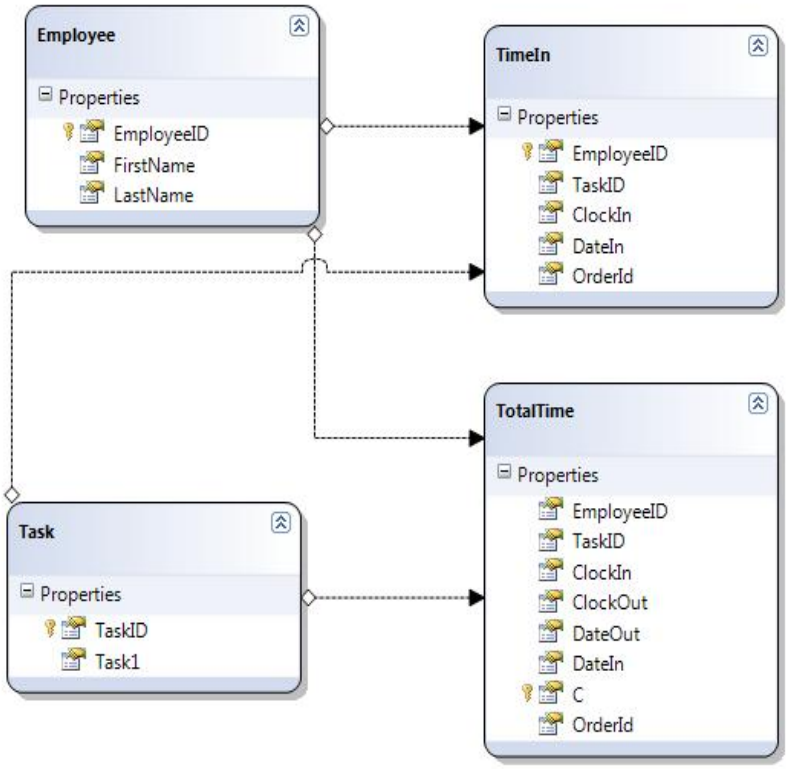
Ramarna runt fönstret är borttagna och fönstret är sedan satt till att vara lika stort som hela skärmupplösningen. Detta ger användaren en känsla av att det enbart är i webb-applikationen man jobbar och att det är det enda programmet som körs på datorn.

En annan fördel, förutom att det körs över hela skärmen, är att tangentbordet går att använda fullt ut, då det inte är själva silverlightapplikationen som körs i fullskärm (se kapitel 2.1.2 och 3.2.3).

Eftersom ingen ram finns runt själva webb-läsaren finns det bara två sätt att stänga ner webbläsaren på, antingen kan man använda tangentbords-kombinationen ALT+F4 eller så kan man klicka på en knapp uppe till höger som är osynlig. I programmet som används nu av System Andersson AB finns ett liknande sätt att stänga ner programmet på och det är genom att klicka på datumet.

4.4 Databas

Databasen som gjordes till applikationen fungerar bra och har de nödvändiga tabellerna och relationerna (se figur 11). Vid interaktion med testapplikationen registreras data korrekt i databasen för att sedan kunna fungera som rapportmall.



Figur 11. Databas med tabeller och relationer.

4.5 Användartester

Användartesterna blev tyvärr inte gjorda som det var tänkt från början då inget företag som använder System Andersson AB:s nuvarande system kunde medverka på tester av mjukvaran. För att få en fingervisning om det är dåligt designat valdes tre studenter med varierande datorvana ut för att testa programmet. Alla tre användarna utförde uppgifterna bra utan problem inom rimlig tid. Alla hittade rätt användare direkt att klicka på. När ordernummer skulle skrivas in var det två användare som hade problem som var av olika karaktär. Den manliga studenten som hade problem skrev in fel ordernummer men rättade detta själv och kan ses som godtagbart och som ett tecken på bra design när han utan problem själv hittade knappen ”rensa” för att börja om. Den kvinnliga studenten fick problem som berodde på att det stod ”ordernummer” på instruktionsblanketten och ”order” i programmet. Detta bör ses över. Alla användare kunde stämpla ut utan problem. Tiden det tog för användarna att använda programmet var ungefär densamma för alla. Det tog bara ett par sekunder att hitta rätt knapp i början och ca 30 sekunder att knappa in rätt ordernummer och stämpla in. Stämpla ut tog också bara ett par sekunder.

Resultatet av testningen får ses som att programmet har en bra design men att man kanske ska se över vad texten till inmatningsfälten säger till användaren. Designen uppskattades av användarna och de relativt enkla uppgifterna som skulle göras gjordes bra och snabbt trots att testpersonerna aldrig hade sett programmet innan eller fått instruktioner på hur det används.

5 Slutsats och diskussion

Vår slutprodukt blev en väl fungerande testapplikation gjord med Microsoft Silverlight. De mål vi hade satt upp för projektet tycker vi uppfylldes helt då detta var en undersökande utredning och vi fick svar de frågor och funderingar som System Andersson AB och vi själva hade ställt. Det blev många positiva svar men även ett negativt vilket var att Crystal Reports i dagsläget inte fungerar ihop med Microsoft Silverlight.

Vi hade satt upp en tidsplan för vårat projekt som inte direkt följdes. Vi stötte på problem då vi skulle göra kopplingen mellan Microsoft Silverlight och Microsoft SQL Server 2008. Vår plan var från början att använda OLEDB-koppling mellan applikationen och databasen men detta fungerade inte utan man var tvungen att använda LINQ.

I början på projektet stötte vi på lite mindre problem som t.ex. hur man skulle konstruera en databas i Microsoft SQL Server 2008 samt att vi behövde bekanta oss med Microsoft Silverlight och Microsoft Expression Blend. Men efter hand som projektet fortlöpte blev dessa problem färre och gick smidigare och snabbare att lösa.

Då vi hade att göra med relativt enkla data som text, tid och datum så var valet av System Andersson AB att använda Microsoft SQL Server 2008 ett bra alternativ för vår applikation.

Vi hade velat göra användartester på personal från ett företag som i dagsläget använder verkstadssystem från System Andersson AB då dessa förmodligen hade genererat lite bättre resultat, men det gick inte att få tag i något företag som kunde ställa upp. Så istället fick vi använda av oss studenter på Högskolan i Jönköping. Vi tycker att testerna blev mycket lyckade då vi fick god respons om användbarhet och design.

Kurser som t.ex. Databasteknik, Systemutveckling, Interaction Design och Programmeringsmetoder har varit till stor hjälp under detta examensarbete både plannering- och utvecklingsmässigt.

Vi anser att det är fullt möjligt att göra det som System Andersson AB är ute efter, vilket är att flytta sin nuvarande windowsbaserade programvara till att bli en webbaserad programvara. Det ända som inte fungerade var rapportskrivining via Crystal Reports men där finns alternativ till detta program som fungerar ihop med Microsoft Silverlight.

Vi tycker att det har varit väldigt roligt och intressant att arbeta med detta projekt. Det har varit utmanande och väldigt lärorikt. Det var kul att få prova på att göra ett projekt där vi fick använda kunskap från tidigare kurser som vi läst i skolan. Vi fick värdefull erfarenhet av att jobba iterativt och söka efter ny information och lösa problem efter hand.

6 Referenser

[1] (u.d.). Hämtat från Microsoft Silverlight: http://msdn.microsoft.com/en-us/library/cc838158(vs.95).aspx den 04 Mars 2009

[2] (u.d.). Hämtat från Microsoft .NET Framwork: http://msdn.microsoft.com/sv-e/library/zw4w595w(en-us).aspx den 05 Mars 2009

[3] (u.d.). Hämtat från Microsoft C#: http://msdn.microsoft.com/sv-se/library/kx37x362(en-us).aspx den 07 Mars 2009

[4] (u.d.). Hämtat från Crystal Reports: http://msdn.microsoft.com/en-us/library/ms225592.aspx den 23 April 2009

[5] (u.d.). Hämtat från Microsoft Visual Basic: http://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx den 16 April 2009

[6] (u.d.). Hämtat från Mono Project: http://mono-project.com/FAQ:_General den 16 April 2009

[7] (u.d.). Hämtat från Language-Integrated Query: http://msdn.microsoft.com/en-us/vcsharp/aa336760.aspx#WhereSimple1 den 18 April 2009

[8] (u.d.). Hämtat från Language-Integrated Query: http://msdn.microsoft.com/sv-se/library/bb308959(en-us).aspx den 24 April 2009

[9] Connolly Thomas, B. C. *Database systems. A practical approach to design, implementation and management* . ISBN 0-321-21025-5.

[10] David, S. I. *An introduction to programming using visual basic 2008 7th edition.* ISBN 978-0-13-066072.

[11] *Interaction design, beyond human-computer interaction, 2nd Edition.* ISBN 978-0-470-01866-8.

[12] Jan, S. (2008). *C++ Direkt.* Studentlitteratur, ISBN 978-91-44-01463-0.

[13] Jan, S. *Skarp programmering med C#.* ISBN 978-9144-05260-1.

[14] Ramez Elmasri, S. B. (2007). *Fundamentals of Database systems 5th edition.* ISBN 0-321-36957-2.

[15] Eriksson Ulf, (2007). Kravhantering för IT-system. ISBN 978-91-44-04728-7

7 Sökord

8 Bilagor

Bilaga 1 Kravspecifikation
Bilaga 2 Databasscript
Bilaga 3 Klassen Employee
Bilaga 4 Service-filen med databasfrågor.
Bilaga 5 Användartest och protokoll

System Andersson AB

Kravspecifikation WEB <-> WIN

Saso Kitanoski DK06
Hannes Zügner DK06

I Innehåll

1.1 Projektbeskrivning

Denna kravspecifikation är en del av examensarbete som görs i samarbete med System Andersson AB som är baserat i Jönköping. Företagen har idag ett verkstadssystem som är utvecklat för att användas i Microsoft Windowsmiljö. Detta system skall testas i en webb-miljö istället och därför behövs det en prototyp för att utreda om det är möjligt att implementera en sådan lösning.
Syftet med denna kravspecifikation är att beskriva funktionerna som prototypen skall ha samt vad som är avgränsat. Detta för att underlätta för alla intressenter att förstå vad det är som egentligen utvecklas. Detta dokument skall även ligga till grund för utvecklarna så att de vet vad det är som skall utvecklas och implementeras.

1.2 Mål

Målet med prototypen är:

- Testa om en webb-applikation är möjlig att använda
- Kan man använda pekskärm
- Går det att köra på annan plattform än microsoft windows på klientsidan
- Går det att köra vilken webbläsare som helst?
- Kan man få bra utskrifter

1.3 Användare

Användarna som kommer använda prototypen kommer vara studenter vid JTH och anställda på System Andersson AB. Dessa användare har god datorvana. Protoypen kommer även att testas på slutanvändare.

1.4 Avgränsningar

Eftersom detta är en prototyp för att testa möjligheten att använda ett system i webb-miljö kommer prototypen vara kraftig begränsad. Det som kommer att implementeras är följande:

- Påstämpling och avstämpling med notering för tid och maskin
- Byta sida i applikationen
- Visa lista över anställningsnummer

1.5 Krav

Funktionella krav
Webbapplikationen har ett antal funktionella krav som behövs implementeras. Dessa krav har diskuterats fram i samspråk med System Andersson AB.

- **Stämpla in**
  För att kunna välja jobb måste användaren stämpla in.

- **Tid/-maskinregistrering**
  Tid för instämpling ska registreras. Val av maskin ska även det kunna registreras.

- **Bläddra sida**
  Det skall finnas möjlighet att bläddra en sida för att undersöka hur detta fungerar.

Utskrifter

- **Crystal report**
  Systemet ska kunna hantera utskrifter från crystal report.

Prestanda

- **Pekskärm**
  Webbapplikationen ska kunna hanteras med pekskärm.

- **Webbläsarens ramar**
  Ramarna till webbläsaren ska kunna styras bort.

- **Streckkodsläsare**
  Systemet ska kunna klara av att använda streckkodsläsare.

DATABASSCRIPT

```
USE [JAJA]
GO
/****** Object:  User [Exjobb]    Script Date: 06/04/2009 23:46:28
******/
CREATE USER [Exjobb] FOR LOGIN [Exjobb] WITH DEFAULT_SCHEMA=[dbo]
GO
/****** Object:  Table [dbo].[Tasks]    Script Date: 06/04/2009
23:46:28 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Tasks](
          [TaskID] [int] NOT NULL,
          [Task] [char](10) NULL,
PRIMARY KEY CLUSTERED
(
          [TaskID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[Employees]    Script Date: 06/04/2009
23:46:28 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[Employees](
          [EmployeeID] [int] NOT NULL,
          [FirstName] [char](10) NULL,
          [LastName] [char](10) NULL,
PRIMARY KEY CLUSTERED
(
          [EmployeeID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
) ON [PRIMARY]
GO
SET ANSI_PADDING OFF
GO
/****** Object:  Table [dbo].[TotalTime]    Script Date: 06/04/2009
23:46:28 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TotalTime](
          [EmployeeID] [int] NULL,
          [TaskID] [int] NULL,
```

```
            [ClockIn] [time](7) NULL,
            [ClockOut] [time](7) NULL,
            [DateOut] [date] NULL,
            [C] [uniqueidentifier] NOT NULL,
            [OrderId] [int] NULL,
            [DateIn] [date] NULL,
PRIMARY KEY CLUSTERED
(
            [C] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  Table [dbo].[TimeIn]    Script Date: 06/04/2009
23:46:28 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[TimeIn](
            [EmployeeID] [int] NOT NULL,
            [TaskID] [int] NULL,
            [ClockIn] [time](7) NULL,
            [DateIn] [date] NULL,
            [OrderId] [int] NULL,
PRIMARY KEY CLUSTERED
(
            [EmployeeID] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  = ON, ALLOW_PAGE_LOCKS  = ON)
ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object:  ForeignKey [FK__TimeIn__Employee__09DE7BCC]
Script Date: 06/04/2009 23:46:28 ******/
ALTER TABLE [dbo].[TimeIn]  WITH CHECK ADD FOREIGN KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO
/****** Object:  ForeignKey [FK__TimeIn__TaskID__0AD2A005]    Script
Date: 06/04/2009 23:46:28 ******/
ALTER TABLE [dbo].[TimeIn]  WITH CHECK ADD FOREIGN KEY([TaskID])
REFERENCES [dbo].[Tasks] ([TaskID])
GO
/****** Object:  ForeignKey [FK__TotalTime__Emplo__0BC6C43E]
Script Date: 06/04/2009 23:46:28 ******/
ALTER TABLE [dbo].[TotalTime]  WITH CHECK ADD FOREIGN
KEY([EmployeeID])
REFERENCES [dbo].[Employees] ([EmployeeID])
GO
/****** Object:  ForeignKey [FK__TotalTime__TaskI__0CBAE877]
Script Date: 06/04/2009 23:46:28 ******/
ALTER TABLE [dbo].[TotalTime]  WITH CHECK ADD FOREIGN KEY([TaskID])
REFERENCES [dbo].[Tasks] ([TaskID])
GO
```

Klassen Employee

```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Ink;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace prototyp3
{
    public class emp
    {
        private bool inStamp = false;
        private string firstName = "";
        private string lastName = "";
        private int employeeId = 0;

        public emp()
        {

        }


        public void setFirstName(string inFirstName)
        {
            //Set firstname
            firstName = inFirstName;
        }

        public string getFirstName()
        {
            //return firstname
            return firstName;
        }

        public void setLastName(string inLastName)
        {
            //set lastname
            lastName = inLastName;
        }

        public string getLastName()
        {
            //return lastname
            return lastName;
        }

        public void setInStamp()
        {
            //set inStamp to true (clock in)
            inStamp = true;
        }

        public void setOutStamp()
        {
            //set inStamp to false (clock out)
```

```
            inStamp = false;
        }

        public bool getStamp()
        {
            //return inStamp
            return inStamp;
        }

        public void setEmployeeId(int inId)
        {
            //set employeeID
            employeeId = inId;
        }

        public int getEmployeeId()
        {
            //return employeeID
            return employeeId;
        }


    }

}
```

Servicefil med databasfrågor

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace prototyp3.Web
{
    // NOTE: If you change the class name "Service1" here, you must
also update the reference to "Service1" in Web.config.
    public class Service1 : IService1
    {

        #region GetEmployeeByFirstName

        public List<Employee> GetEmployeeByFirstName(string fn)
        {
            // Gets employees from employees by first name

            DataClasses1DataContext db = new
            DataClasses1DataContext();
            var matchingEmployees = from empList in db.Employees
                                     where
                                empList.FirstName.StartsWith(fn)
                                   select empList;
            return matchingEmployees.ToList();

        }

        #endregion

        #region GetEmployeeByEmployeeID


        public List<Employee> GetEmployeeByEmployeeID(int eId)
        {
            //Gets employees from employee by epmloyeeid

            DataClasses1DataContext db = new
DataClasses1DataContext();
            var matchingEmployees = from empList in db.Employees
                                    where empList.EmployeeID < eId
                                    select empList;
            return matchingEmployees.ToList();
        }

        #endregion

        #region insertIntoTimeIn

        public void insertIntoTimeIn(int empId, int tId, int
        orderNumber)
        {
            //Inserts time into TimeIn

            DataClasses1DataContext db = new
DataClasses1DataContext();
            DateTime current = DateTime.Now;
```

```
        // Create a new row object.
        TimeIn row = new TimeIn
        {
            EmployeeID = empId,
            TaskID = tId,
            ClockIn = current.TimeOfDay,
            DateIn = current.Date,
            OrderId = orderNumber
        };
        // Add the new object to the collection.
        db.TimeIns.InsertOnSubmit(row);

        // Submit the change to the database.
        db.SubmitChanges();
    }

    #endregion

    #region getTaskByEmployeeID

    public List<TimeIn> getTaskByEmployeeID(int empId)
    {
        //Get task by EmployeeID

        DataClasses1DataContext db = new
        DataClasses1DataContext();
        var matchingStampIn = from stamp in db.TimeIns
                              where stamp.EmployeeID == empId
                              select stamp;
        return matchingStampIn.ToList();
    }

    #endregion

    #region deleteFromTimeIn


    public void deleteFromTimeIn(int empId)
    {
        //Deletes rows from TímeIn when no longer in use

        DataClasses1DataContext db = new
        DataClasses1DataContext();
        var deleteStamp = from stamp in db.TimeIns
                          where stamp.EmployeeID == empId
                          select stamp;
        db.TimeIns.DeleteOnSubmit(deleteStamp.First());

        db.SubmitChanges();
    }

    #endregion

    #region insertIntoTotalTime


    public void insertIntoTotalTime(List<TimeIn> listTaskIn, int
    empId, int tIn)
    {
        //Gets a list with timein and inserts into totaltime
```

```
            DataClasses1DataContext db = new
            DataClasses1DataContext();
            DateTime current = DateTime.Now;
            // Create a new row object.
            TotalTime row = new TotalTime
            {
                C = Guid.NewGuid(),
                EmployeeID = empId,
                TaskID = tIn,
                ClockIn = listTaskIn.First().ClockIn,
                DateIn = listTaskIn.First().DateIn,
                ClockOut = current.TimeOfDay,
                DateOut = current.Date,
                OrderId = listTaskIn.First().OrderId
            };
            // Add the new object to the collection.
            db.TotalTimes.InsertOnSubmit(row);

            // Submit the change to the database.
            db.SubmitChanges();
        }

        #endregion

        #region getTaskByTaskID


        public List<Task> getTaskByTaskID(int tId)
        {
           DataClasses1DataContext db = new
           DataClasses1DataContext();
           var matchingTasks = from taskList in db.Tasks
                               where taskList.TaskID < tId
                               select taskList;
            return matchingTasks.ToList();
        }
        #endregion
    }
}
```

Användartest och protokoll

Användartester – Man 25 år och student vid Tekniska Högskolan i Jönköping, mycket god datorvana.

1. Hitta namn Sebastian Öwall.
2. Stämpla in med ordernummer ”10063” och operation ”borra”.
3. Stämpla ut namn Sebastian Öwall.

Kommentarer från användare

Bra: Går väl inte att göra lättare än såhär? Lätt att hitta namn.

Dåligt:

Övrigt: Hittade knappen ”rensa” vid num-paden direkt då fel siffra slagits in.

Protokoll – Man 25 år och student vid Tekniska Högskolan i Jönköping, mycket god datorvana.

1. Hittade namn inom 3 sek. Gjordes några fel? Nej.
   Kommentar, Uppskattning/Missnöje:

2. Stämpla in med ordernummer och operation inom 26 sek. Gjordes några fel? Nej.
   Kommentar, Uppskattning/Missnöje:

   Användaren hittade knappen ”rensa” direkt då fel siffra slagits in.

3. Stämpla ut inom 3 sek. Gjordes några fel?
   Kommentar, Uppskattning/Missnöje:

Användartester – Kvinna 22 år och student vid Internationella Handelshögskolan i Jönköping, mindre god datorvana.

1. Hitta namn Marie Björk.
2. Stämpla in med ordernummer ”10052” och operation ”Slipa”.
3. Stämpla ut namn Marie Björk.

Kommentarer från användare

Bra: Väldigt enkelt

Dåligt:

Övrigt: Det kunde stått ordernummer istället för bara order, tog lite tid att förstå.

Protokol – Kvinna 22 år och student vid Internationella Handelshögskolan i Jönköping, mindre god datorvana.

1. Hittade namn inom 3 sek. Gjordes några fel? Nej.
   Kommentar, Uppskattning/Missnöje:

2. Stämpla in med ordernummer och operation inom 31 sek. Gjordes några fel? Nej.
   Kommentar, Uppskattning/Missnöje:

   Användaren funderade lite över vart ordernummer skulle skrivas in.

3. Stämpla ut inom 3 sek. Gjordes några fel?
   Kommentar, Uppskattning/Missnöje:

Användartester – Man 27 år och student vid Tekniska Högskolan i Jönköping, mycket god datorvana.

1. Hitta namn Magnus Öien.
2. Stämpla in med ordernummer ”10041” och operation ”Såga”.
3. Stämpla ut namn Magnus Öien.

Kommentarer från användare

Bra: Snyggt och enkelt interface. Stora och tydliga knappar.

Dåligt:

Övrigt: Väl fungerande och praktiskt.

Protokol – Man 27 år och student vid Tekniska Högskolan i Jönköping, mycket god datorvana.

1. Hittade namn inom 4 sek. Gjordes några fel? Nej.
   Kommentar, Uppskattning/Missnöje:

2. Stämpla in med ordernummer och operation inom 21 sek. Gjordes några fel? Nej.
   Kommentar, Uppskattning/Missnöje:

   Användaren hade inga problem att navigera i systemet.

3. Stämpla ut inom 4 sek. Gjordes några fel?
   Kommentar, Uppskattning/Missnöje: