Datavetenskap

Magnus Langered och Denny Pedersen

THEO tidsregistrering

Examensarbete, C-nivå

2000:16

THEO tidsregistrering

Magnus Langered och Denny Pedersen

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

Magnus Langered och Denny Pedersen

Godkänd, 2000-05-29

Handledare: Börje Arvidsson

Examinator: Stefan Lindskog

Sammanfattning

I denna C-uppsats redogör vi för arbetet att revidera ett tidsrapporteringsprogram (THEO-tidsregistrering) ägt av IT-företaget Benefit Partner AB. Programmet skulle revideras därför att det inte alltid fungerade som det skulle och nya önskemål angående funktionalitet hade uppkommit.

Programmet är skrivet i Visual Basic och använder en Oracle-databas för informationslagring.

När vi skrev om programmet rensade vi först den ursprungliga koden från kodrader som inte längre användes, såsom gammal testkod. Under denna rensning fick vi även en första överblick av hur programmet fungerade. Därefter skrevs det mesta av koden om för att få ett stabilare, snabbare och bättre skrivet program.
Den främsta skillnaden i den nya versionen gentemot den tidigare är att funktionerna har blivit fler men generellare och mindre. Koden har optimerats så att ingenting längre körs i onödan och många inmatningstester och felkontroller har kunnat tas bort då inmatningen blivit mer restriktiv och felkällor har eliminerats.

Det problem vi främst stött på under arbetets gång har varit att den kod och databas vi hade som utgångsmaterial nästan helt saknade kommentarer och förklaringar.

Resultat är ett program som är snabbare, kortare och stabilare. Källkoden är lättläst, logisk och dokumenterad, vilket medför att programmet kommer vara mycket lättare att förstå för eventuell vidareutveckling.

THEO time-registration

Abstract

The purpose of this bachelor's project was to update a time-report system (THEO- time-registration) owned by the company Benefit Partner AB. The program needed an update because of unreliable execution and new ideas concerning the functionality.

The program is written in Visual Basic and uses an Oracle database to store information.

When we rewrote the program, we first cleared the original source code from parts that was no longer in use. This gave us a first overview of how the program worked. After that, most of the source code was rewritten to get a more stable, faster and an overall better program.
The foremost difference in the new version compared to the previous one is that the number of functions has increased but they are more generalized and not as big in size. The source code has been optimized so no code executes in vain and many input tests and error controls have been removed. This is due to more restrictive input and elimination of error sources.

The main problems we have encountered during this project have been due to lack of comments and explanations of the source code and the database.

The result is a program that is faster, shorter, more stable and nicer. The source code is readable, logic and well documented, which will make it easier to understand for further development.

Innehållsförteckning

Figurförteckning

Appendix A:

1 Inledning

1.1 Bakgrund

IT-företaget Benefit Partner AB använder sig idag av THEO, ett egenutvecklat system som bland annat används för löneberäkning och projektstatistik. Systemet är skrivet i programspråket Visual Basic och använder en ODBC-koppling (Open DataBase Connectivety) till en Oracle databas för informationslagring. Dessutom används ActiveReport för att generera rapportutskrifter.

THEO lagrar information om vilka projekt som finns, vilka aktiviteter som finns under varje projekt, vilka anställda som finns, vilka som är projektledare, arbetad tid mm. Varje projektledare tilldelar de anställda på olika aktiviteter inom projektet. En databasbild över de tabeller som tidsregistreringsdelen använder sig av finns i Appendix A.1.

Det gamla systemet motsvarade inte den av Benefit framtagna kravspecifikationen. THEO består idag av åtta delar men vi hade som uppgift att endast förbättra tidsregisteringsdelen. Detta är den del av THEO där alla anställda lägger in den tid de har lagt ner på olika projekt och deras aktiviteter. Eftersom denna del skall användas dagligen av samtliga på Benefit var det här som det var störst behov av en tillförlitlig och fungerande applikation.

1.2 Problembeskrivning

Första intrycket av THEO:s tidsregistreringsdel gav en känsla av att systemet var instabilt. Det fungerade inte riktigt som konventionella programvaror brukar göra. Inmatning i rutorna fungerade inte såsom i Microsoft Excel, till exempel kunde inte delete, return och piltangenterna användas när man stod i en inmatningsruta i rutnätet. För varje inmatning man gjorde i respektive ruta räknades summor för alla dagarna ut trots att endast den dag som ändrats behövde räknas ut. Detta tog onödigt lång tid och gav ett osäkert intryck då det ”fladdrade” till överallt i cellerna.

När man behövde byta månad eller år, ändrades storleken på rutnäten fram och tillbaka.
Vissa småfunktioner i programmet fungerade inte alls. Utskriftsrapporterna för projektuppföljning och tidsredovisning fungerade inte som de skulle, systemet låste sig,

dokumentationen av kod och system var så gott som obefintlig och ibland lade systemet till felaktig information.

1.3 Avgränsning

Databasens uppbyggnad exkluderades för att säkerställa att arbetet skulle hinna avslutas inom avsatt tid (20 veckors studier på halvfart).

1.4 Syfte och mål

Vårt syfte och mål har varit att inom tidsramen av 20 veckor tillhandahålla IT-företaget Benefit Partner AB en tillförlitlig och uppdaterad version av applikationen THEO tidsregistrering.

1.5 Tillvägagångssätt

Tillvägagångssättet har varit att först lära oss Visual Basic och sedan sätta upp en utvecklingsmiljö. Därefter analyserades koden och slutligen skrevs programmet om.

2 Lösning

2.1 Antaganden

Då vi inte hade någon detaljerad kravspecifikation att utgå ifrån har vi antagit att programmet överlag skall fungera som det gjorde innan, men med vissa förändringar. Vid inmatningsförfarandet har vi infört ett mer excel-liknande uppförande. Vi har även gjort inmatningen mer restriktiv och det går till exempel inte att mata in bokstäver i tidsrutorna. Funktionalitet som aldrig användes togs bort och vi lade till lite mer färger för att få ett trevligare utseende.

2.2 Konstruktionsbeskrivning

Under konstruktionen av programmet strävade vi efter att få så enkla, korta, snabba och få funktioner som möjligt. Detta framförallt för att få programmet stabilt och lätt att testa.

Programmet kan sägas var uppbyggt i fem steg:

- Uppkoppling
- Grundläggande initiering
- Uppdatering under körning
- Lagring i databasen
- Avslutning

Nedan följer beskrivning av de olika stegen.

Uppkoppling

När programmet startar skapas en ADO-koppling (ActiveX Data Objects, manipulerar data genom en ODBC- brygga [2]) till databasen. Denna databaskoppling ligger uppe så länge som programmet körs och används av alla de SQL-kommandon som finns i koden för att hämta och spara information.

Grundläggande initiering

Efter uppkoppling körs funktioner som endast behöver köras en enda gång. Exempel på detta är placering av grafiska objekt i användargränssnittet för aktuell skärmupplösning, formatering och utplacering av vissa textsträngar och dimensionering av rutnäten.

Uppdatering under körning

När den grundläggande initieringen har kört färdigt anropas de funktioner som kör varje gång som år eller månad ändras. Dessa funktioner fyller i vilka projektaktiviteter som den inloggade har fyllt i timmar för, fyller i de timmar som arbetats under respektive datum och räknar ut alla totalsummor.

De rutnät som finns i Visual Basic går inte att editera under programkörning. För att simulera detta läggs en textruta över den ruta man vill editera och inmatad text förs sedan över till rutnätet. Detta syns inte ut mot användaren och är en standardlösning för att få editering i rutnät [2].

För att slippa ta hand om en massa felinmatningar har programmet konstruerats så att endast giltiga värden går att skriva in.

Lagring i databasen

För att hålla reda på vad som ändras när en person fyller i timmar i programmet har en speciell strukt (egendefinierad datatyp) skapats. Denna strukt håller reda på rutnät, projekt, aktivitet, kolumn och ändringstyp för den ändring som gjorts. Det finns tre typer av ändringar, *insert*, *update* och *delete*, som alla kräver olika SQL-kommandon för att uppdatera databasen. Ändringarna sparas allt eftersom de skapas i en dynamisk array och när man sedan väljer att spara stegas arrayen igenom och databasen uppdateras med en ändring i taget. När hela arrayen är avklarad töms den.

Avslutning

Vid avslutning kontrolleras om allt är sparat. Därefter tas databaskopplingen ner och programmet avslutas.

2.3 Metod

När vi hade bekantat oss med Visual Basic var nästa steg att sätta upp en utvecklingsmiljö. Denna bestod av en kopierad databas (detta för att inte förstöra redan lagrad information), en ODBC-koppling till testdatabasen, utvecklingsverktyget Visual Basic 6.0 med servicepack 3, en Oracle-klient samt det befintliga tidsregistreringsprogrammet. Det gjordes även säkerhetskopior på källkoden.

Nästa steg var att analysera koden. Vid första anblicken av koden förstod vi inte mycket då den var ganska omfattande och hade väldigt mycket ”bortkommenterad” kod i sig, d v s kod som hade inaktiverats och inte längre var körbar. I koden fanns även många siffervärden som inte var förklarade, så kallade ”magic numbers”. Kommentarer om vad funktioner och variabler var till för var det däremot ont om. Namnen på variablerna var i många fall kryptiska och det var inte självklart vad de användes till.

Detta var bland annat ett resultat av dålig dokumentation och att programmet hade uppdaterats ett antal gånger av olika personer. För att kunna få någon sorts överblick över källkoden började vi med att rensa bort alla ”bortkommenterade” kodrader.

I koden fanns många variabler som var kvar från tidigare programversioner och variabler som använts för felsökning. Vi kontrollerade därför alla variabler för att se vilka som aldrig användes och tog bort dessa. Det fanns också variabler som tilldelades värden men som aldrig

användes. Dessa variabler och kodrader var ganska många och kodmängden kunde minskas relativt mycket. Uppskattningsvis hittade vi ca 40 helt onödiga variabler med tillhörande kod. De variabler som blev kvar typdeklarerades då detta inte alltid var gjort innan. Efter denna första ”kodrensning” blev koden något mer lättläst men långt ifrån överskådlig.

Nästa steg blev att förenkla de funktioner som fanns. Dessa kunde fås att göra samma sak men på ett enklare och smidigare sätt. Detta gav även en bra översikt över hur programmet fungerade och var uppbyggt.

När vi hade kommit så här långt hade vi fått en grundläggande inblick i hur programmet fungerade men än var inte allt helt klart. Vi gick vidare med att rita upp ”kodflödet” i programmet och vi förbättrade och skrev om funktioner allt eftersom de anropades.

Efter att på detta sätt gått igenom programmet hade vi bra inblick i hur det fungerade och kunde nu börja med att optimera programmet. Vi hade märkt att många kodrader kördes i onödan. Det räckte att köra dem en gång men de låg i funktioner som behövde köras ofta, dessa kodrader lyftes ut till egna funktioner som endast anropas en gång. Funktionerna gjordes kortare och snabbare med så få variabler som möjligt. Funktionerna gjordes även mer generella så att de kunde användas på många ställen och koddubbleringar eliminerades.

Exempel på ändringar i programmet:

- Dialogrutorna ”Tips” och ”Om” gjordes modala. Dessa två dialogrutor finns under menyn Hjälp. Se Appendix A.2 Figur 2.5 och Figur 2.6. (En modal dialogruta innebär att man som användare måste stänga, genom att klicka OK eller Avbryt innan han eller hon kan fortsätta med någon annan del i programmet). Detta har fördelen att man slipper ha ett antal olika fönster öppna under programmets körning och dessutom färre komponenter att hålla reda på. Detta för i sin tur med sig att det är färre saker som kan gå fel.
- En funktion som var cirka tio till elva sidor lång, kortades ner till cirka en sida kod. Funktionen gjorde samma sak tolv gånger med skillnad endast i en ”integer” och en ”sträng”, detta kunde lätt läggas i en ”loop”.
- En egengjord funktion som användes för att byta ut delsträngar i en sträng (t.ex. byta ut ”.” mot ”,” i strängen ”10.5”) byttes ut mot den fördefinierade funktionen *Replace*. Denna funktion är ny för Visual Basic 6.0.

- Det fanns en funktion som användes för att summera alla dagar i det övre rutnätet. Då detta inte är den bästa lösningen gjordes två stycken nya funktioner. Funktionen *Dagsumma* som används för att räkna ut antalet timmar man har jobbat en viss dag, och funktionen *Radsumma* som räknar ut totala antalet timmar för en projektaktivitet under en månad.

Förändringarna innebar att koden blev mycket mer lättläst.

2.4 Uppnådda förbättringar

- Programmet är snabbare, mindre och uppdaterar stabilt.
- Koden är mer strukturerad och uppdelad i småfunktioner.
- Kod körs inte i onödan.
- Inmatningen är säkrare och enklare.
- Rapporterna är mer strukturerade.
- Småbuggarna som fanns tidigare är borta.
- Funktionerna och variablerna är dokumenterade.
- SQL-satserna är mer specifika.

2.5 Installation

Installation av THEO tidsregistrering går till enligt följande:

1. En klient för Oracle läggs in på datorn och konfigureras. Detta behövs för att kunna komma åt databasen.
2. ODBC-koppling till databasen skapas. Denna koppling behövs då programmet anropar databasen med denna teknik.
3. En installationsfil för THEO tidsregistrering körs. Denna lägger till alla dll-filer (Dynamic Link Libraries, filer som innehåller rutiner för Visual Basic program [1]) och ActiveX-filer (en ActiveX komponent är en programspråksoberoende komponent [1]) som behövs samt exe-filen för programmet.

3 Diskussion

3.1 Problem

3.1.1 Händelsestyrd programmering

Ett program skrivet i Visual Basic är händelsestyrt. Programmet förväntar sig att man som användare skall utföra någon form av händelse eller handling. Denna händelse kan bestå av ett klick, ett dubbelklick eller en tangentbordstryckning som programmet skall svara på.

Visual Basic har inbyggda kontroller som fångar upp dessa händelser. Dock måste en händelseprocedur implementeras för att den skall kunna utföra avsedd funktion.

Ett problem har varit att en händelse kan generera flera olika händelseprocedurer. Om man arbetar med cellhantering och vill ställa sig i ny cell krävs ett ”musklick” eller en tangentbordstryckning. Denna procedur genererar händelserna: *MouseDown*, *MousePress*, *MouseUp* (för musen), *LeaveCell* och *LostFocus* för cellen man stod i och till sist *EnterCell* och *GotFocus* för den nya cellen. Denna händelseordning har inte varit helt självklar. Då det mest logiska skulle ha varit att *LostFocus* skulle ha anropats efter klickhändelsen och inte *LeaveCell* och *EnterCell*. Detta förfarande ställde till en hel del problem vad gäller editeringen i rutnäten, dessutom kunde en ny cell markeras utan att *EnterCell* anropades.
Ett annat problem var att man kan generera händelser direkt i källkod utan att själva händelsen anropas. Detta kan vara bra om man har varit med och tagit fram programmet från början, man vet *var* och *varför* en viss händelseprocedur utförs. Men det är väldigt svårt att förstå *var* och *varför* en sådan händelseprocedur utförs om man letar efter buggar.

3.1.2 Generell struktur

Ett problem var att databasen och programmet inte alltid hade en överensstämmande struktur. När man lagrade namnen på aktiviteterna i det nedre rutnätet (se Appendix A.2 Figur 2.3), hade de annorlunda namn i själva databasen. Detta gjorde att man var tvungen att ha flera ”extra” funktioner. Det kunde ha ordnats genom att uppdatera aktivitetsnamnen för det nedre rutnätet i databasen, men det hade inneburit att all lagrad data också hade behövt ändrats.

3.1.3 Visual Basic och databasen

Man kan inte säga att Visual Basic är precisionssäkert när det gäller omdefiniering av redan satta storlekar på grafiska objekt. Det gick inte att styra exakt position, vilket gjorde att hela rutnäten inte syntes i programmet.

Ett problem som uppstod efter man avslutat programmet var att det fortfarande kördes i minnet. Detta löstes med kommandot *End* som är ett ”hårt” sätt att avsluta program.

När man i kod bytte aktivt rutnät kunde man i Visual Basic sätta en låst ruta (i rutnätet) markerad. Med låst ruta menas att den rutan inte skall vara editerbar. Efter bytet kunde rutan editeras trots att det fanns funktioner som skulle förhindra detta.

Flera problem uppstod kring databashanteringen. Om man till exempel gick in i databasen och ändrade lagrad data med hjälp av SQL-satser och sedan startade THEO tidsregistrering, syntes inte ändringarna i programmet. Problemet uppstod eftersom kommandot *commit* (data i databasen lagras permanent) inte körts.

Ett annat problem var att vissa fält i databasen innehöll ogiltiga värden trots att tabellerna i databasen var deklarerade på ett sådant sätt att detta skulle vara omöjligt.

I det ursprungliga programmet skedde alla databasanrop via en RDO-koppling (Remote Data Objects, en äldre typ av databaskoppling).
Decimaldelen på decimaltal lagrades inte i databasen med RDO-koppling, men när vi bytte till ADO fungerade det, varför det inte fungerade med RDO vet vi ej men ADO är en nyare typ av uppkoppling och vi hoppas att detta även löser de problem som ”gamla” THEO tidsregistrering har med databasen.

3.1.4 Installationen

Theo tidsregistrering fungerar alldeles utmärkt vid skärmupplösningarna 1024x768 och 800x600. Men när skärmupplösningen blir mindre än 800x600 så ser användargränssnittet inte så bra ut eftersom det då inte får plats på skärmen och delar överlappar.

THEO tidsregistrering fungerar under operativsystemen Windows NT/98 men det har uppkommit vissa problem då man försöker installera programmet på en dator med Windows 95.

Eftersom THEO tidsregistrering till stor del handlar om att lagra data ett visst datum så har en del problem uppstått då man kan välja olika datumformat i operativsystemet. Det datumformat som vi valt att lagra/visa datum på är YYYY-MM-DD (t.ex. 2000-05-04), men beroende på vilken språkversion man använder så uppstår problem om man lagrar/visar datum i andra format, till exempel YY-MMM-DD (t.ex. 00-Maj-04).

3.2 Förslag till förbättringar

- Feltester vid databasanrop som testar om allt gått rätt till kan läggas till. Detta är en viktig förändring och borde göras så fort som möjligt.
- Man skulle kunna göra ett mer genomarbetad användargränssnitt som klarar fler skärmupplösningar.
- Gränsnittet för programmet kan göras snyggare med egen design på knappar osv.
- Bankrutan (se Appendix A.2 figur 2.3 i högra hörnet) med semestertimmar och flextimmar fungerar inte i dagsläget men borde kunna fås att fungera. Att den inte fungerar idag beror på bristfällig databasdesign och att man inte riktigt vet vad den skall visa.
- Eftersom vi endast har arbetat med tidsregistreringen i THEO bör man se över de andra sju delarna som hör till systemet. Dessa delar har alla en koppling till varandra och fel i en del kan leda till fel i en annan del.

3.3 Alternativa lösningar

Alternativa lösningar kunde ha varit att göra tidsregistreringsdelen i THEO webbaserad genom att använda servlets (javaprogram som körs på en server) eller ASP (Active Server Pages, dynamiska websidor). Det hade då bland annat inte behövts någon programinstallation.

Ett annat alternativ vore att skriva om THEO tidsregistreringsdel i ett annat programspråk. Exempelvis C++ för att få programmet snabbare.

Som programmet fungerar idag sker uppdatering till databasen med ett kommando (*insert*, *update* och *delete*) i taget, men detta skulle kunna göras om så att alla uppdateringar skickas på samma gång.

3.4 Erfarenheter och slutsatser

3.4.1 Dokumentation

Att dokumentera det man har gjort är oerhört viktigt för att kunna ge andra personer möjlighet att sätta sig in i programmet på ett relativt enkelt sätt för eventuella framtida uppdateringar. Dokumentationen till THEO tidsregistrering med avseende på databasen och källkoden, har i stort sett varit obefintlig.

Det har krävts mycket arbete att förstå *hur* och *varför* vissa funktioner i källkoden används.

3.4.2 Övrigt

- Ge bra namn på variabler och funktioner i källkoden. Det är oerhört svårt att förstå och sätta sig in i en uppgift om man inte har bra variabelnamn och funktionsnamn. Namngivningen av variabler och funktioner bör vara konsekvent och ha en förklarande innebörd av dess uppgift. När program blir riktigt stora är detta ännu viktigare.
- Skriv korta funktioner. En funktion bör vara *högst* en sida lång. Blir funktionen längre än en sida bör man dela upp den i mindre funktioner. Detta för att få en mera överskådlig och lättläst kod över vad programmet gör. Dessutom är det bättre att konstruera små och korta funktioner för att få programmet mer modulärt vilket underlättar eventuella uppdateringar.
- Ta bort ”bortkommenterad” kod. Det är bättre att ta bort kod som inte gör något annat än att ligga kvar och ”skräpa”. Det hela blir annars oöverskådligt.
- Online hjälp är utmärkt som referenslitteratur. Den kan dessutom ge vägledning om hur man kan lösa ett problem.

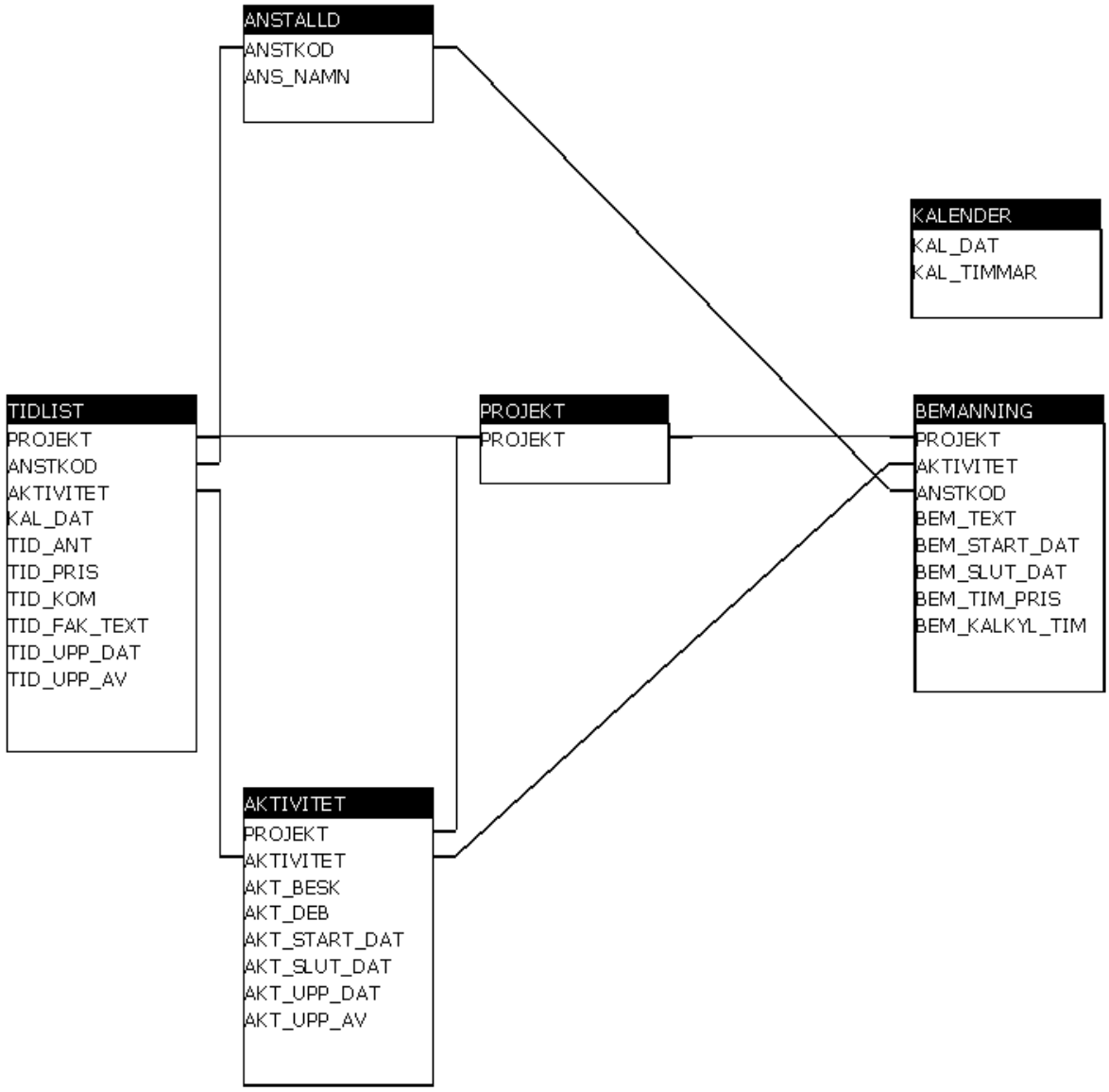
4 Litteraturförteckning

[1] *G. Perry, Lär dig Visual Basic 6 på 3 veckor, Pagina Förlags AB, 1999.*

[2] *MicroSofts Online Hjälp, MSDN Library*

A Appendix

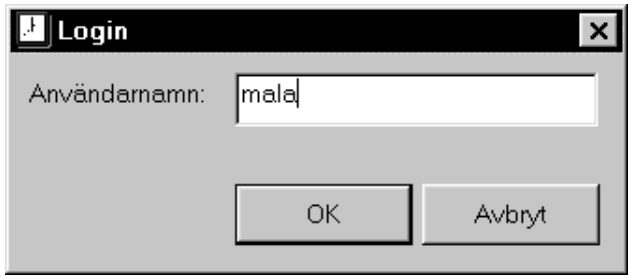
A.1 THEO 2-tidsregistrerings databas.



Figur 1.1: Tidsregistreringens del av databasen.

A.2 Programbeskrivning.

Det första man gör när man vill använda THEO tidsregistrering är att logga in med sitt användarnamn. Se figur 2.1.

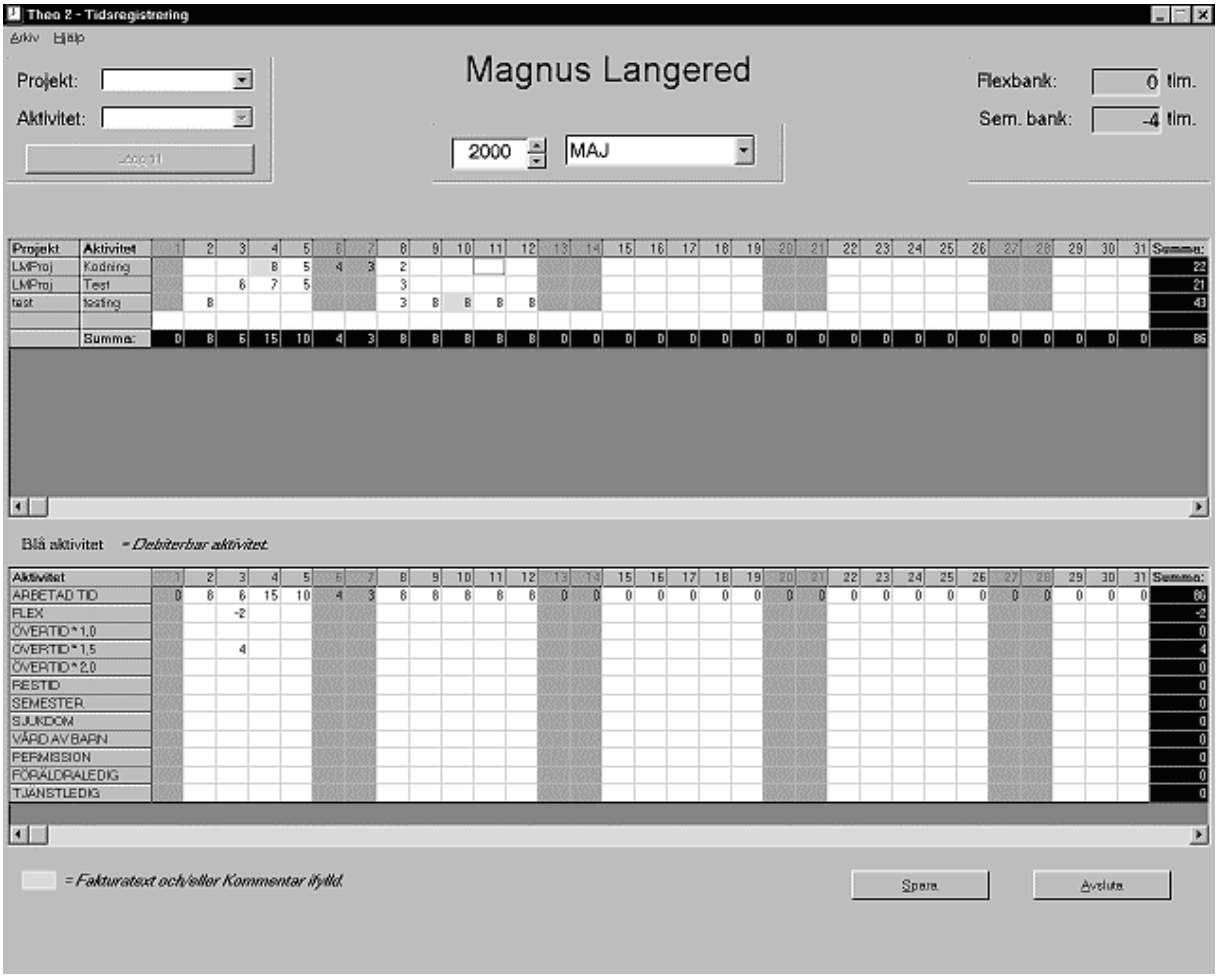


Figur 2.1: Inloggningsformulär.

Är användarnamnet giltigt så möts man av en splash screen. Se figur 2.2. En splashscreen är en inledningsbild som visas innan programmet startar.

Figur 2.2: Exempel på splash screen.

Om användarnamnet är giltigt d v s du finns registerad i databasen startar THEO tidsregistrering upp. Se figur 2.3.



Figur 2.3: Theo 2 - Tidsregistrering.

Skulle inte användarnamnet vara giltigt återkommer man till inloggningsformuläret, där man kan ange användarnamnet på nytt.

THEO tidsregistrering består av två stycken menyer, tre stycken comboboxar (val/alternativ), fjorton stycken labels (visar en textsträng), tre stycken knappar, en UpDown kontroll och två stycken rutnät.

Under Arkivmenyn finns en undermeny *Utskrift* samt alternativen *Spara* och *Avsluta*.

Utskriftsmenyn består av *Projektrapport* och *Tidrapport*. Se figur 2.4.

Figur 2.4: Utskriftsmenyn.

-Projektrapporten är en utskrift för projektledaren där det står hur man som anställd har fördelat sin arbetstid i olika projekt och aktiviteter. Se exempel Appendix A.4. Projektrapport.

-Tidrapporten är en utskrift för löneadministratören för beräkning av löner. Se exempel Appendix A.5. Tidrapport.

Spara alternativet sparar hela rutnäten för aktuell månad i databasen.

Avsluta alternativet avslutar hela programmet.

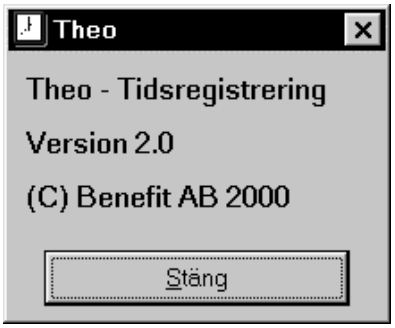
Under Hjälpmenyn finns alternativen *Tips* och *Om*. Se figur 2.5. Detta är två stycken formulär.

Figur 2.5: Hjälpmenyn.

-*Tips* visar värdefulla tips som kan användas i THEO tidsregistrering. Se figur 2.6.

Figur 2.6: Tips.

-*Om* visar programversion och copyright. Se figur 2.7.

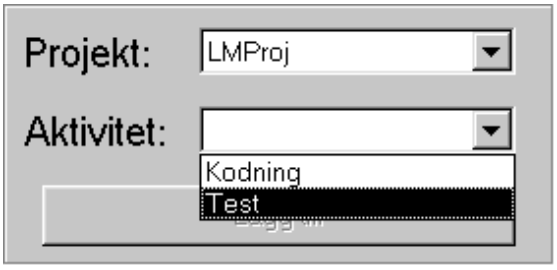


Figur 2.7: Om.

De tre comboboxar som finns är Projekt, Aktivitet och Månad.

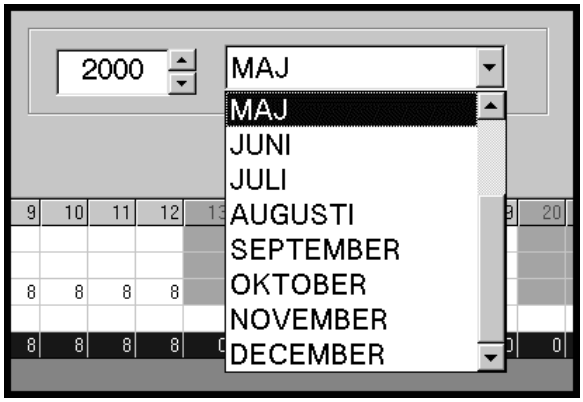
-Projekt visar vilka projekt som man är bemannad på.

-Aktivitet visar vilka aktiviteter som finns under valt projekt så länge de inte finns inlagda i övre rutnätet. Se figur 2.8.



Figur 2.8: Projekt och aktivitet.

-Månad där väljer man vilken månad man vill visa. Se figur 2.9.



Figur 2.9: Månadsalternativ.

De labels som finns används för att visa användarens namn och övrig text som förekommer.

De knappar som finns är Lägg till, Spara och Avsluta.

-”Lägg till”-knappen används för att lägga till ett projekt och dess tillhörande aktivitet i det övre rutnätet.

-”Spara”-knappen används för att spara ändringar man gjort i rutnäten.

-”Avsluta”-knappen används för att avsluta hela programmet.

UpDown används för att byta år.

Det finns två stycken rutnät. Det övre och undre rutnätet.

-Det övre rutnätet används för att visa hur arbetad tid är fördelad över olika projekts aktiviteter.

-Det undre rutnätet redovisar arbetad tid för löneberäkningar.

I rutnäten markeras arbetsfria dagar med en gråfärgning.

När programmet startar upp visas alltid aktuellt år och månad. Rutnätet fylls med tidigare inlagd information som hämtas upp från databasen.

För att lägga till en aktivitet i det övre rutnätet väljs först ett projekt och sedan en aktivitet under det tillhörande projektet. Därefter trycker man på knappen ”Lägg till” som infogar och sorterar aktiviteten i det övre rutnätet. Man kan genom att klicka på ”summafältet” för en viss aktivitet få fram hur många timmar man arbetat och hur många timmar aktiviteten är kalkylerad till. Se figur 2.10.

Figur 2.10: Aktivitetsrapport.

Om man vill kan man vid registering av tid för ett visst datum infoga en kommentar eller fakturatext för just det datumet genom att dubbelklicka i cellen man önskar fylla i. Se figur 2.11.

Figur 2.11: Inskrivning av fakturatext och /eller kommentar.

A.3 Projektrapport

Projektrapport för: Magnus Langered

MAJ 2000

Debiterbara Projekt	**Aktivitet**	**Summa**
LMProj	Kodning	22
test	testing	43
		65

Interna Projekt	**Aktivitet**	**Summa**
LMProj	Test	21
		21

Totalt
86

A.4 Tidrapport

Tidrapport för: Magnus Langered

MAJ 2000

Aktivitet	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	Summa:
ARBETAD TID	0	8	6	15	10	4	3	8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86
FLEX			-2																													-2
ÖVERTID*1,0																																0
ÖVERTID*1,5			4																													4
ÖVERTID*2,0																																0
RESTID																																0
SEMESTER																																0
SJUKDOM																																0
VÅRD AV BARN																																0
PERMISSION																																0
FÖRÄLDRALEDIG																																0
TJÄNSTLEDIG																																0