Linnéuniversitetet
Institutionen för datavetenskap, fysik och matematik

Examensarbete

Till framtida utvecklare

Jimmie Yngvesson
2011-05-31
Ämne: Datavetenskap
Nivå: B
Kurskod: 1DV40E

Abstrakt

Syftet med rapporten är hur man på bästa sätt kan utveckla en applikation som ska lämnas över till en framtida utvecklare.
För att undersöka detta har ett utvecklingsarbete skett hos en verklig kund där arbetet i framtiden kan fortsätta med annan utvecklare.
Ett viktigt fokus under arbetets gång har varit att tänka sig in i den situation som en framtida utvecklare kan tänkas hamna i.

Jag har valt att fokusera på val av verktyg, kodstandard och arkitektur och hur dessa överförs till kommande utvecklare.
Resultatet av rapporten visar att mycket tid och resurser kan besparas om man tänker sig in i en framtida utvecklares situation.

Abstract

The purpose of the report, how best to develop an application that will be distributed to prospective developers.
To examine this, a development occurred in an actual customer where the work will continue with another developer.
An important focus during the work has to constantly keep themselves in the situation that a future developer might end up in.
I have chosen to focus on the choice of tools, control over the architecture and how they are transmitted to future developers.
The result of this report is that a lot of time and resources can be spared if one imagines in a future developer's situation.

Förord

Arbetet är gjort som ett avslutande examensarbete i programmet Webbprogrammerare 120 poäng vid Linnéuniversitetet i Kalmar.

Uppdraget kom från ett enskilt företag som ville ha en helt ny site för att visa upp och ha försäljning av utvalda produkter. Jag ville direkt från början utveckla mina kunskaper i de programmeringsspråk som jag ansåg jag behövde mer i.

Jag vill rikta ett riktigt stort tack till Daniel Toll som varit handledare under arbetet med rapporten.

Innehållsförteckning

1. Introduktion 1

2. Bakgrund 2

3. Metod 2

3.1 Metoddiskussion 2

4. Utvecklingsmiljö 3

4.1 Hur väljer man en miljö till en framtida utvecklare? 3

4.2 Lösning i uppdraget mot kund 4

4.2.1 Val av programvara 4

4.2.2 Överlämning av programvara 4

5. Källkod 6

5.1 Hur ska man hantera och lagra källkod som ska lämnas över 6

5.2 Lösningen i uppdraget mot kund 7

5.2.1 Val av lagringsteknik 7

5.2.2 Versionshanteringsprogram - överlämning 8

5.2.3 Lagring av källkod - Val av tredje part leverantör 8

5.3 Exempel på medskickade instruktioner 9

6. Arkitektur 10

6.1 Hur väljer man arkitektur som ska lämnas över 10

6.2 Lösningen i uppdraget mot kund 11

6.2.1 Val av teknik 11

6.2.2 Överföra arkitektur till någon annan 11

6.3 Exempel på medskickad beskrivning av arkitektur 12

7. Kodstandard 14

7.1 Hur man väljer kodstandard 14

7.2 Lösningen i uppdraget mot kund 15

7.2.1 Val av kodstandard 15

7.3 Exempel på medskickad beskrivande kodstandard 16

8. Slutsats 17

9. Källförteckning 18

9.1 Elektroniska källor 18

1. Introduktion

Rapporten grundar sig på ett examensarbete vid det tvååriga webbprogrammeringsprogrammet vid Linnéuniversitetet i Kalmar. Syftet med arbetet har varit att ta fram en webbplats åt en enskild firma som säljer och installerar pooler och spa-bad. Webbplatsen ska förutom att visa information även erbjuda en webbshop på utvalda produkter. Då tid och kunskap inte från början var tillräckliga klargjordes det från början att vissa delar inte skulle bli klara, därför kommer kunden att behöva hjälp utav en framtida utvecklare för att fortsätta projektet.

Rapporten kommer innehålla bland annat:

- Vilka allmänna problem kan en framtida utvecklare ställas inför?
- Saker som kan bli ett problem med just min applikation
- Beskrivning hur jag löst problemen
- Detaljerade exempel på hur man underlättar ett överlämnande

2.Bakgrund

Ett problem idag är när personer får i uppdrag att vidareutveckla applikationer som andra tidigare har gjort. En situation är att man får hela projektet med enbart källkod och inget mer. Vid många fall kommer man inte att träffa eller kunna få tag i de personer som har gjort arbetet innan. Här är det givetvis önskvärt att när man börjar vidareutveckla förstå vilka tekniker som använts, vilken utvecklingsmiljö man bör använda och var man hittar befintlig information om applikationen. Listan kan göras väldigt lång. Vad detta innebär i praktiken är att väldigt mycket onödig tid läggs på att bara komma igång med projektet. Denna tidskrävande procedur kostar självklart onödiga pengar, som säkert hade kunnat användas på bättre sätt.

Med utgång av detta problem ska jag med denna rapport visa hur man som utvecklare borde hantera sina projekt för att framtidens utvecklare kan göra ett så bra arbete som möjligt.

3. Metod

Jag har valt att fokusera på utvecklingsmiljö, källkod, arkitektur och kodstandard.För att fundera på hur man bäst lämnar över arbetet har en applikation tagits fram. Under tiden jag utvecklat applikationen har jag funderat på hur olika beslut skall dokumenteras och överlämnas.

Jag kommer skapa överlämningsdokument som innehåller instruktioner och exempel men även källkod och programvaror som kommer att behövas om man fortsätter programmeringsarbetet.

3.1 Metoddiskussion

För att eventuellt få till ett mer tillförlitligt resultat, borde mer undersökning kring varje punkt göras. Som exempel borde jag mer detaljerat gjort undersökningar om vilka programvaror som kan vara lämpliga att jobba med, om tanken är att lämna över arbetet. Hade omständigheterna tillåtit så hade olika test mot andra utvecklare kunnat göras för att få en bättre bild av olika resultat. Detta hade kunnat vara att man gjort mindre överlämningar av de beskrivningar med mera som jag gjort, för att se vilket resultat detta kunnat ge. Mitt metodval kan sammanfattas med att jag inte hade någon framtida utvecklare att lämna över till efter avslutat arbete, vilket då resulterar i att jag inte hade den kommunikation när det gällde val av saker att fokusera på i min metod.

4 Utvecklingsmiljö

4.1 Utvecklingsmiljö - hur väljer man en miljö som kan lämnas över till andra utvecklare?

Idag finns det otaliga program och hjälpmedel som är till för att hjälpa utvecklare i sitt dagliga arbete.

Den som tar över ett projekt vet inte vilka program och verktyg som använts, eller varför. Det som jag ställt i fokus är hur utomstående utvecklare ska få samma utvecklingsmiljö som jag har haft. Detta antagande har jag själv gjort, där jag försökt tänka mig in i en sådan situation där jag själv ska ta över ett påbörjat projekt. Att få igång samma utvecklingsmiljö som tidigare utvecklare är mer effektivt än att själv börja ladda hem eller installera andra programvaror som ska fungera ihop med påbörjat projekt.

Vad bör man tänka på när man väljer vilka verktyg man ska jobba med? Här dyker frågor som pris, säkerhet, support, stabilitet, enkelhet, datorprestanda, plattform upp. Ska man vidareutveckla en befintlig applikation är det bra att veta vilka verktyg som använts tidigare. Problem finns även när det krävs licens för att utveckla applikationer. Vid större projekt med många inblandade blir kostanden en viktig faktor, och även den tid det tar att ordna de licenser som behövs.

I en del fall så är en del av programmen gjorda för enbart vissa plattformer som Windows eller Mac OS, vilket kan resultera i att man inte kan köra ett visst program på sin dator. Man bör givetvis se över vad det är för sorts applikation det är man ska göra, då vissa program inte kan skapa alla sorters kod.

Hur har man det med support eller eget kunnande om något skulle krångla? Detta är ju en annan viktig sak att ta i beaktning. Det är inte alltid de mest välkända namnen som är enklast att komma igång med, eller har bäst dokumentation. Många gånger kan man helt enkelt inte ens lyckas att installera programmet då något oväntat inträffat. Att göra sökningar på olika forum där problem och lösningar finns är en väldigt bra start att börja med för att få en generell uppfattning om vad marknaden har att erbjuda.

En lista på många av de mest använda texteditorerna kan hittas på denna länk: http://en.wikipedia.org/wiki/Comparison_of_text_editors, där även kostnad bland annat går att utläsa.

4.2 Utvecklingsmiljö - lösning i uppdraget mot kund

4.2.1 Val av programvara

Projektet är skrivet med Aptana Studios 2. Detta beror på att det är en editor som jag blivit van med, den är gratis och kompatibel med den kod som jag skrivit till applikationen. Jag har valt att använda mig av version 2 då detta är en officiell testad version. Detta innebär att om man i framtiden ska utveckla applikationen, och det är fler personer som ska jobba med den behöver man inte heller tänka på att ordna licenser.

För att utveckla databasen har programmet MySqlworkbench använts. Jag jämförde med det mer vanliga PhpMyAdmin och hittade ett par fördelar med just Workbench. En av de största fördelarna är att man kan sitta och skapa databasen lokalt på sin egna dator, för att sedan överföra via programmet till den skarpa databasen. Andra fördelar med programmet är att det finns många fler valmöjligheter att jobba med som diagram, relationer mellan tabeller och att det går lätt skapa lagrade procedurer. Även detta är ett gratis program och information och aktiva forum gör att man snabbt kan få hjälp och information om man behöver.

4.2.2 Överlämning av programvara

För att se till att en annan utvecklare ska få samma förutsättningar och samma miljö som jag har haft ska detta lösas genom att skicka med alla de program som har använts i projektet. Detta i form av körbara programfiler som startar installationen, och då även enkla instruktioner hur man genomför detta. Detta är som sagt till för att göra det enklare och snabbare för en annan utvecklare komma igång med samma förutsättningar som jag har, och personen i fråga slipper att själv testa vilka program som fungerar bäst ihop med det projekt jag påbörjat.

Samma sak gäller för eventuella tillägg som måste göras. Jag har även tänkt på att skicka med information angående att programmen regelbundet byter version. Det som kan inträffa är att den version jag medskickat inte stämmer överens med dom nödvändiga plugins som eventuellt finns när en annan utvecklare ska sätta upp sin

miljö. Resultatet av att skicka med alla nödvändiga filer och dokumentation gör att man lätt kan komma igång med sin miljö och snabbare komma igång med att arbeta. Ett exempel på medskickad instruktion är när jag utvecklade databasen lokalt och sedan skulle överföra arbetet till den skarpa databasen som då ligger hos Binero. Här visade det sig att man måste använda en tredje parts program och göra vissa manuella inställningar för att få det att fungera.

4.2.2.1 Exempel på installations instruktioner

Här visas ett exempel på instruktion vid installation av plugin i Aptana:

Viktigt!

För att kunna utveckla PHP-kod i Aptana måste man installera ett plugin som gör detta möjligt. Följ instruktionerna nedan:

1. När installationen av Aptana är färdig och man startat programmet syns en flik som heter "My studio" längst upp. Även olika val presenteras i startfönstret när "my studio" fliken är vald
2. I startfönstret finns det ett alternativ som heter "Install plugins" vilket man ska klicka på
3. I nästa vy presenteras 4 olika menyval, här ska man välja "Featured"
4. Flera val dyker upp och det som ska väljas här är: "PHP Development Tools"
5. Bekräfta dina val och följ guiden för att installera
6. Starta om Aptana om detta efterfrågas
7. Klart.

Denna information bör självklart skickas med då det kommer underlätta mycket åt den som ska jobba med det.

5. Källkod

5.1 Källkod - Hur ska man hantera och lagra källkod som ska lämnas över?

Hantering av källkod kan se ut på många olika sätt. Man kan ha det lokalt på sin egen dator, på en egen server eller hos en tredjepartsleverantör.

Jag måste då sätta mig in i den situation som en framtida utvecklare kommer att ha då min källkod ska lämnas över. Förmedling av de val som gjorts angående lagring och hanteringen av källkod ska göras på ett förståligt sätt.

Ett av problemen är när många ska arbeta med samma applikation. Då arbetar man antagligen flera samtidigt på samma saker, vilket snabbt kan bli förvirrande då man inte vet vilka ändningar som gjorts av vem, eller vilken version som är den senaste. En annan sak som kan inträffa är att lagringsmediet man har valt att spara sitt arbete på går sönder eller på något sätt inte går att få tag på. Utan backup är detta ingen trevlig situation att råka ut för. Ett annat scenario kan ju vara att en person själv har utvecklat viktiga klasser i ett projekt, och sedan sparat sitt arbete på sin privata dator eller lagringsmedia. Vad händer om personen i fråga blir sjuk eller på något annat sätt inte kan nås?

Det jag syftar på i detta fall är versionshantering. Problem uppstår om personen som ska vidareutveckla aldrig arbetat med versionshantering. Begrepp som: Subversion, TortoiseSVN, Google Code, är saker som en framtida utvecklare eventuellt aldrig kommit i kontakt med.

Ett annat fokus är att förklara hur jag arbetat med att lagra all viktig information hos en tredje part. Saker som är viktiga att förklara är bland annat säkerhet, hanteringen av

befintliga konton, hur man kommer igång med de grundläggande funktionerna med mera.

Andra problem som kan uppstå kan vara att den som tar över har en annan plattform än den som projektet är utvecklat i. De program som då följt med kan då ställa till problem. Här kommer vikten att läggas på att tydligt demonstrera de val som gjorts för att hantera dessa problem. Jag måste förutsätta att den som tar vid inte jobbat enligt den modell jag har gjort

5.2 Källkod - lösning i uppdraget mot kund

5.2.1 Val av lagringsteknik

Hela projektet har versionshanterats, vilket gör det viktigt att generellt beskriva vilka fördelar som detta kan ge.

Varför ska man då versionshantera? Ett starkt argument för detta är att om flera olika personer editerar och ändrar i filer blir det lätt och snabbt kaos i ett vanligt filsystem eftersom det blir väldigt svårt att veta vilken version som är den senaste. Det som ett versionshanteringssystem hjälper till med är man enbart kan ändra i en fil som är av senaste version. Andra finesser är att man kan gå tillbaka till äldre versioner, se vad som är ändrat av vem och så vidare.

Med detta i åtanke bör viktiga filer och dokument sparas i ett versionshanteringssystem. Det finns även här många olika versionshanteringssystem som: Subversion, Git, Clearcase. För en mer detaljerad lista av system besök följande länk: http://sv.wikipedia.org/wiki/Versionshantering.

Till dessa versionshanteringssystem finns det olika verktyg att använda sig av. Några av dem är TortoiseSVN, RapidSVN. Fler finns att läsa om på följande länk: http://sv.wikipedia.org/wiki/Apache_Subversion. Vid val av versionshanteringssystem får man tänka på vilka plattformar dom är utvecklade för. Som exempel är TortoiseSVN utvecklat enbart för Windows.

Mitt val av versionshanteringssystem föll på Subversion och med verktyget TortoiseSVN. Avgörande punkter var att jag jobbat med det sedan tidigare vilket innebar att inlärningen inte behövdes. Det är ett fritt program att använda och det finns gott om dokumentation generellt om programmet. Själva nedladdningen av programmet kan göras via följande länk: http://tortoisesvn.net/downloads.html.

5.2.2 Versionshanteringsprogram - överlämning

Körbara program har även här skickats med för att komma igång med versionshateringsprogrammen.

För att få en så kort inlärningskurva som möjligt medföljer kortfattade beskrivningar som rör installation av programmen, beskrivning med exempel som visar på hur man dagligen jobbar med dessa, samt de eventuella problem som upptäcktes. Ett exempel på detta är mjukvaruklienten TortoiseSVN som enbart är anpassat till Windows. Blir situationen den att den som vidareutvecklar gör detta på Mac OS istället så kommer inte det medskickade programmet att fungera

Information om alternativa program bör även följa med, då det inte alls är säkert att de som tar över har samma plattform som projektet har utvecklats i.

5.2.3 Lagring av källkod - val av tredjepartleverantör

Då kunden inte har någon egen dedikerad server för lagring av filer med backup har jag valt att lägga all källkod hos en tredje part.

Här har enkelhet, säkerhet, pris, och inlärningskurva i beräkningarna. Då Google Code sedan tidigare är bekant så blev valet ganska enkelt och snabbt. Det är ett snabbt, gratis och säkert alternativ att komma igång med, och då inte tid fanns att göra jämförelser så blev valet detta.

Förutom mina argument varför valet föll på Google Code, så blir det viktigt att förmedla vidare hur detta ska hanteras. Tillräcklig kunskap måste medfölja projektet, eftersom det inte är säkert att den som tar över har arbetat med en tredjepartsleverantör tidigare. Saker som bör behandlas är: Säkerhet, hantering av konton, beskrivning hur man kommer igång med Google Code. Det är viktigt att förklara att lösenord, användarnamn och andra personliga uppgifter inte bör lagras hos Google Code då det ändå är en publik server man använder. Sådana uppgifter som rör befintliga konton som skapats under projektet ska därför följa med direkt till den eller dom som ska utveckla vidare. Även här ska det tydligt framgå var dessa uppgifter finns sparade, att man har en detaljerad beskrivning hur man ska hantera konton och på vilket sätt man kommer igång med Google Code.

5.3 Exempel på medskickade instruktioner

Nedan listar jag ett par exempel som visar hur medskickad information ser ut.

Exempel 1: Hur man dagligen bör jobba med TortoiseSVN

1. Gör en Update på antingen en hel katalog eller en specifik fil, detta gör att du får hem den senaste versionen av filen. Ha som vana att göra update ofta.

2. Arbeta med filen och gör nödvändiga förändringar och testa så att allt funkar

3. Gör en Commit på antingen en hel katalog eller en specifik fil, detta gör att du publicerar dina förändringar så att andra projektmedlemmar kan få hem dem när de gör Update. Du kan även skriva in en kommentar om vilka förändringar du gjort.

Ju oftare man gör detta desto bättre, det vill säga många små förändringar är bättre än en jättestor.
När du gör en Commit finns en chans att någon annan har gjort en Commit innan dig, det vill säga du arbetar inte i den senaste versionen av filen. Du måste då göra en Update innan du gör en Commit, Tortoise försöker då slå samman dina förändringar med den senaste versionen av filen, detta fungerar oftast mycket bra. Dock kan man i vissa fall själv få välja vilka förändringar man vill ha i filen, man har en så kallad Conflict som man måste åtgärda innan man kan göra en Commit. Med en bra struktur på sitt respositorium, bra arbetsfördelning och om man kör Update ofta kan risken för konflikter minimeras.
Hjälpfilen till TortoiseSVN är mycket omfattande och rekommenderas

Exempel 2: Hur du ansluter till Google Code, och kommer åt projektet

1. Besök följande länk : http://code.google.com/

2. Här ska man logga in i meny valet som finns uppe till vänster

3. Här ska följade inloggnings uppgifter matas in.

E-post: E-post för inloggning finns hos ansvarig person

Lösenord: Lösenord finns hos ansvarig person

4. I menyvalet uppe till höger klickar man på "My favorites" och sedan på webbprojekt

5. Du kommer nu åt hela projektet där inblandade personer finns, vilka uppladdningar som gjort, och även vilka ändringar som är gjorda av vem och när. Även andra

menyval finns tillgängliga för att kunna jobba och justera olika inställningar som rör projektet.

6. När man väl är inloggad finns det en "getting started" länk som visar de olika funktionerna och tips hur man kan jobba med Google Code. Denna sida är väl värt att lägga lite tid och läsa igenom

6 Arkitektur

6.1 Arkitektur - hur väljer man en arkitektur som ska förstås av andra när det lämnas över?

Idag bygger man applikationer med varierande tankesätt och tekniker. Detta kan som exempel vara att man gör funktionsbaserade eller objektorienterade applikationer. Man kan välja att göra en applikation i enbart flash eller så faller valet på att man ska följa standarden för HTML 5. Här blir alternativen återigen många och detta kan ställa till problem för framtida utveckling.

Fokus blir att utgå från att en framtida utvecklare inte alls har samma synsätt på att bygga upp en arkitektur. Att överföra min syn och tankesätt vidare till de som ska arbeta vidare med applikationen blir viktigt.

Ett exempel på problem som kan dyka upp är om mitt projekt ska vidareutvecklas av personer som har för vana att göra det med en funktionsbaserad kod. Hur ska jag på ett så enkelt sätt som möjligt överföra en annan teknik att bygga applikationer på?

Utan bra instruktioner hur min arkitektur är tänkt att fungera så blir detta en väldigt tidsödande process för de som ska arbeta vidare med projektet.

Ett annat problem är att många tolkar vissa tekniker olika. Model View Controller, mer känd som MVC är en sådan teknik som ofta används vid utveckling av diverse applikationer. Grundtanken med tekniken är ju att separera de olika lagren för att på så sätt få en renare och enklare kod att underhålla, och att får en separation mellan logik och design. Några bra länkar som beskriver tekniken bakom MVC finns här http://www.phpportalen.net/article.php?id=12

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Problem uppkommer dock ändå då vi tolkar saker på olika sätt.

Hur ska jag förmedla arkitektur, tolkningar av tekniker, namngivning, projektstruktur med mera?

Utan tydliga förklaringar och exempel som visar hur man konkret har tänkt så blir det även här en lång och tidskrävande process för en framtida utvecklare att sätta sig in i.

Strukturen på hela projektet är även det en viktig del att tänka på. Att lämna över ett projekt utan någon som helst struktur innebär en massa tid att reda ut vilka filer och dokument som tillhör vad.

6.2 Arkitektur - lösning i uppdraget mot kund

6.2.1 Val av teknik

Då jag under min studietid lärt mig att skriva objektorienterat, och använt MVC som modell när vi utvecklat applikationer, så föll valet ganska enkelt på att tillämpa detta på projektet

Hela projektet har byggts upp från grunden och jag har tillämpat en MVC teknik som använts under studietiden. De fördelar som detta innebär är att man får en bra kontroll och översikt på sina filer och man lär sig ganska snabbt vilka filer som tillhör vad i sin struktur.

Andra fördelar är ju att just separationen mellan design och logik blir enkel att hantera. Man ska kunna ändra i vissa delar utan att andra blir påverkade av ändringarna. Ett annat sätt att se på det är att man löser ett stort problem enklare om man delar upp det i små och mindre problem.

6.2.2 Överföra min arkitektur till en annan utvecklare

För att nu se till att en vidareutveckling av applikationen ska kunna gå så smidigt som möjligt överförs beskrivningar, exempel, tankar och tolkningar som jag har haft under projektets gång. En åtgärd har varit att skriva exempel på vissa utvalda delar från applikationen som ska underlätta förståelsen och förhoppningsvis gör att nästa person snabbare kan anamma samma teknik som använts.

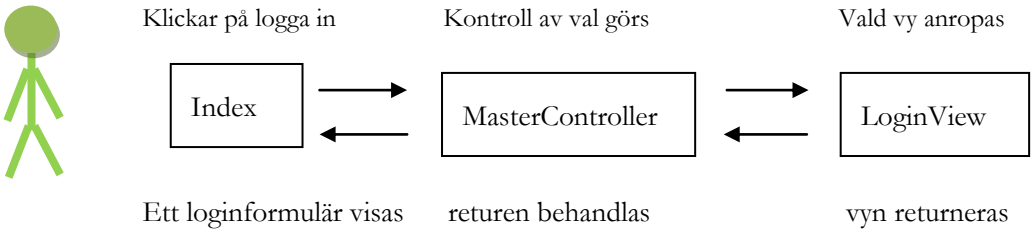
Jag har även tänk på att få över samma tankesätt att framtidssäkra applikationen till nästa utvecklare. Det vill säga att den som tar vid efter mig ska fortsätta i samma spår, och att man poängterar vikten i att tänka vad som kan hända i framtiden. I mitt fall är

hela applikationen gjord i PHP. Som ett open source-språk innebär detta att man bör hålla sig uppdaterad om vilka ändringar och förbättringar som kan tillkomma.

6.3 Exempel av medskickad beskrivande arkitektur

Projektets teknik utgår från att man aldrig lämnar startsidan (index.php) utan jag har byggt applikationen på så sätt att man byter vyer och funktionalitet beroende på vad användaren gör på sidan. Följande exempel beskriver mer i detalj.

Ursprungsläget är att startsidan visas



Jag har valt att skapa en egen klass som mer eller mindre kontrollerar de val man kan göra som användare. Denna klass heter MasterController och det är denna klass som kommunicerar med de andra klasserna beroende vad du som användare gör. Detta gör också att man får en mycket renare och lätthanterlig index.php

Nedan beskrivs ett exempel där en användare vill logga in på sidan. Här får man även en inblick hur MasterControllern fungerar.

1. Då all interaktion går via MasterControllern, är det denna som pratar med de andra klasserna

2. Användaren har på sidan klickat på logga in knappen

3. Kontrollen av detta har gjorts inne i MasterControllern.

4. En variabel tilldelas det värde som kom tillbaka, i detta fall ett inloggnings formulär

5. Variabeln returneras i slutet av funktionen, och resultatet kan beskådas av användaren

För att öka förståelsen ytterligare skickas även ett kodexempel från ovanstående situation med.

```
if($_GET["controller"] == "login")
{
 $xhtml .= $logView->DoLoginForm();
}
return $xhtml
```

Att ha denna övergripande funktion gör att det blir enkelt att lägga till nya kontroller och funktionalitet i MasterControllern.

Att ha en instruktion hur man kan lägga till ny funktionalitet i MasterController bör bifogas och kan se ut enligt följande:

Scenario: Ni vill utöka huvudmenyn med ytterligare ett val

1. Lägg till det nya valet i vyn som innehåller huvudmenyn (MainNavView.php)

2. Ge det nya valet ett passande namn

3. Gå sedan över till Mastercontroller.php

4. I funktionen DoControll() skriver man den kod som kontrollerar om användaren har valt det nya alternativet. Visar exempel nedan

```
if($_GET["controller"] == "namnet som ni gav menyvalet")
{
$xhtml .= //Här skapas anrop som sedan tilldelas $xhtml
}
```

7 Kodstandard

7.1 Hur väljer man en kodstandard som ska förstås av andra utvecklare

När det gäller hur man ska skriva sin kod finns det ju inget som säger att man är tvungen att göra detta på ett visst sätt. Vissa utvecklar sin egen standard, en del följer en rekommenderad, och vissa kanske skriver sin kod efter något ramverk.
Några andra exempel som kan ställa till problem är namngivning på funktioner som många gånger kan hittas på http://www.php.net.

Nedan listas ett axplock av exempel som kan förbrylla en ovan programmerare.
stristr (string $haystack , mixed $needle [, bool $before_needle = false])
strcmp (string $str1 , string $str2)
bcsqrt (string $operand [, int $scale])
bcpow (string $left_operand , string $right_operand [, int $scale])
Vad jag vill påpeka med ovanstående exempel är att man bör följa en enklare standard som möjliggör för utomstående att kunna läsa och förstå den kod du skriver.
För att följa en kodstandard som ändå gäller för den stora mängden, gäller det att ta en titt på hur marknaden ser ut idag. Då det gäller utveckling mot ramverk har de ofta en egen standard för hur man bör skriva kod. Ett par exempel på erkända ramverk finns att läsa om på följande länkar: http://www.zend.com/en/ och http://codeigniter.com/. Att tänka på dock om man vill använda ramverk är att det finns olika grader av inlärningskurva på dessa.

Men vilken standard bör man nu följa om man inte använder ett av de större ramverken? Gör man lite enkla sökningar på ämnet ihop med att man vill skiva PHP kod dyker namnet Pear´s upp regelbundet. För att läsa mer vad detta innebär besök följande länk: http://pear.php.net/index.php. Här kan man enkelt få konkreta exempel på hur man bör skriva sin kod. Det innebär allt från hur du bör namnge funktioner till hur man bör placera måsvingarna.

Det man som utvecklare ändå ska tänka på är att man bör välja en standard som man vill skriva efter, och att man sedan håller sig till den. Kan man en brukar det inte vara så svårt om man vill anamma en annan.

7.2 Kodstandard - lösning i uppdraget mot kund

7.2.1 Val av kodstandard

Då erfarenhet från de större ramverken saknas, valde jag att följa Pears kodstandard med ett par undantag. Andra anledningar att valet föll på Pears är att det använts under studietiden, och att det är en bra site att besöka för att få information med mera.
Ett par undantag när det gäller namngivning har gjorts. Pears följer en standard som säger att en klass ska namnges med ett understreck mellan varje ord. Jag har istället använt en princip som heter PascalCase, som innebär att varje nytt påbörjat ord har stor bokstav. Här finns det också alternativ då man kan välja att ordet ska börja med liten bokstav (kallas då camel-case) eller att ordet börjar med stor bokstav (PascalCase)
Man kan läsa mer om detta på följande länk: http://sv.wikipedia.org/wiki/Camelcase

Skillnaden mellan att namnge en klass kan beskrivas med följande exempel:
Pears recommendation:

```
class Foo_Bar
{

    //... code goes here

}
```

Min lösning med PascalCase:

```
class FooBar
{

    //... code goes here
}
```

När det gäller funktionsnamn börjar samtliga med stor bokstav(PascalCase), medans variabler börjar med liten bokstav(camelCase).

7.3 Exempel av beskrivande kodstandard

Då jag avvikit på ett par punkter när det gäller namngivning är det bra att en sådan beskrivning följer med. Ett exempel på en sådan beskrivning kan se ut enligt nedan:

Namngivning av privata variabler.
I vissa klasser förekommer privata medlemsvariabler där jag valt att namngivningen av variabeln börjar med $_m. Detta är en äldre standard som är framtagen från Microsoft från början, och har dykt upp under den tid vi läste PHP kursen.

Så ett exempel för detta hittas i klassen DataBaseConnection där vi har en privat variabel som sparar information om namnet på databasen. Detta ser ut enligt följande: private $_mdatabaseName = "";.
Standarden för Pears är likvärdig, det som skiljer är att då hade man uteslutigt m:et. Med andra ord hade raden sett ut så här: private _dataBaseName.

Det andra problemet som fokus har lagts på är namngivning av funktioner. Många gånger hittar man funktionsnamn som är riktigt usla. Vissa av dessa namn klargjordes i avsnittet hur jag valde kodstandard.

Hur ska man skapa funktionsnamn som innebär att en utomstående ser vad det är som just den funktionen ska uträtta?

För att öka förståelse om vad en funktion ska uträtta har jag arbetat med beskrivande namn på samtliga funktioner. Att visa vilken standard man följt när det gäller namngivning kan underlätta arbetet för en framtida utvecklare.

Exempel på namngivning av funktioner enligt Pears:

```
ShowCustomerDetails($userName)
ListCustomers()
SaveUser($firstName, $lastName, $street, $zipCode, $city, $email,
$phone, $mobile, $userName, $password )
```

Detta borde ge en rätt klar bild om vad dessa funktionerna gör. Ovanstående namngivning är också exempel på hur man följer kodstandarden enligt Pears, med principerna camelCase (variabler) och PascalCase(funktioner).

8.Slutsats

Min slutsats av hela arbetet blir att med hjälp av de instruktioner, beskrivningar och att man bifogar de nödvändiga delar som till exempel mjukvara, drastiskt kommer att minska arbetsbördan för den eller de som ska vidareutveckla. Den tid det tar från det att man får projektet i hand tills det att man kan komma igång att arbeta ska minska betydligt.

I mitt fall hade jag inte den kontakt med kunden som jag hade hoppats på i början. Då kunden inte är den som ska vidareutveckla fick den jag inte heller den feedback man kunde hoppats på.

Vid ett överlämnade finns det ändå en bra grund att jobba vidare på om kunden i framtiden vill vidareutveckla applikationen.

Vad man i framtiden kan undersöka är hur ett verkligt överlämnande av applikationen påverkar en framtida utvecklare. Här hade man fått möjlighet att kontrollera om man reducerar den tid det tar att komma igång med arbetet.

Andra saker man hade kunnat utveckla är automatiska tester som en framtida utvecklare hade kunnat utnyttja för att kunna se hur saker fungerar i källkoden.

Jag hoppas att innehållet i rapporten innebär att alla som utvecklar applikationer av någon form tar åt sig, och tänker till en extra gång på den person som eventuellt ska fortsätta arbetet.

9.Källförteckning

9.1 Elektroniska källor

Wikipedia.(2009) *Comparision of texteditors.*
http://en.wikipedia.org/wiki/Comparison_of_text_editors[2011-05-18]

Wikipedia.(2010) *Versionshantering.* http://sv.wikipedia.org/wiki/Versionshantering
[2011-05-18]

Wikipedia.(2011). *Apache Subversion.* http://sv.wikipedia.org/wiki/Apache_Subversion
[2011-05-18]

TortoiseSVN Download/info http://tortoisesvn.net/downloads.html [2011-05-18]

MVC. *Introduktion av MVC.* http://www.phpportalen.net/article.php?id=12

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

[2011-05-18]

PHP. Information/Forum. http://www.php.net. [2011-05-18]
Zend. PHP framework. http://www.zend.com/en/. [2011-05-18]
Codeigniter. PHP framework. http://codeigniter.com/ . [2011-05-18]
Pear´s. *PHP Extension and Application Repository.* http://pear.php.net/index.php.
[2011-05-18]
Wikipedia. (2011) *Camelcase.* http://sv.wikipedia.org/wiki/Camelcase .[2011-05-18]

Linnéuniversitetet
Institutionen för datavetenskap, fysik och matematik

351 95 Växjö / 391 82 Kalmar
Tel 0772-28 80 00
dfm@lnu.se
Lnu.se