GÖTEBORGS UNIVERSITET

Webbapplikationer, från fristående kod till implementering av ett ramverk

- En studie om vilka komplikationer som kan uppstå när applikationer baserad på fristående kod ska överföras till lämpligt ramverk

Web applications, from standalone code to implementing a framework

- A study of the complications that can occur when applications based on standalone code should be transferred to the appropriate framework

Mårten Nilsson

Kandidatuppsats i informatik

Rapport nr. 2012:031
ISSN: 1651-4769

Göteborgs universitet
Institutionen för tillämpad informationsteknologi
Göteborg, Sverige, Maj 2012

Sammanfattning

Bakgrunden till denna uppsats rör underhåll av mjukvara vilket tar mycket resurser i förhållande till de som används under utvecklingen av ny mjukvara. För att på ett effektivare sätt utveckla och underhålla mjukvara har olika typer av ramverk utvecklats. Det intressanta är om dessa ramverk även kan vara fördelaktiga att använda för redan utvecklade applikationer skrivna i fristående kod. Detta har testats genom att överföra en redan fungerande webbapplikation skriven i fristående kod till ett lämpligt ramverk.

Syftet med uppsatsen är att belysa eventuella komplikationer som kan komma att uppstå när en webbapplikation där befintlig mjukvara kan anses rimlig att överföra till ett passande ramverk. Syftet är även att kunna visa om vidare utveckling och underhåll blir effektivare i valt ramverk, detta i förhållande till överföringen. Vid överföringen autogenererades kod av betydande mängd vilket gjorde att överföringen gick relativt smidigt. För fortsatta uppdateringar och underhåll för överförd applikation var inte fördelarna så påtagliga som väntat i förhållande till överföringen. Utifrån studiens resultat kan det anses som en fördel att överföra en redan utvecklad applikation om det finns ett betydande behov av vidareutveckling och underhåll av applikationen. Studien visade även att det inte finns några direkta hinder för att genomföra en överföring.

Nyckelord: Software framework, Web application framework, webbapplikationer, refactoring, toFramework refactoring.

Förord

Arbetet med denna uppsats har gett mig goda möjligheter att fördjupa mig i de delar av informatikämnet som jag tycker är intressant. Uppsatsen har dessutom varit en del av förbättringen av en produkt som redan finns tillgänglig på marknaden och som har befintliga kunder. Att kunna kombinera dessa två världar är för mig till stor nytta och lärdom.

Jag vill även ta chansen att rikta ett tack till min handledare Lennart Petersson som varit tillgänglig och coachande i sitt stöd genom hela skrivprocessen. Lennart har låtit mig föra uppsatsens i den riktning jag velat men ändå givit stödjande feedback.

Sist vill jag tacka min förstående sambo som inte fått se så mycket av mig de senaste månaderna då jag parallellt med denna uppsats haft flera andra projekt, tack.

Göteborg den 24 Maj, 2012

_____

Mårten Nilsson

Innehållsförteckning

Figur- och tabellförteckning

1 Inledning

I detta kapitel presenteras uppsatsens bakgrund, problemdiskussion, problemformulering, syfte och vidare disposition. Bakgrunden till denna uppsats har sitt ursprung i utvecklingen av mjukvaruutveckling och ramverk anpassade för webbapplikationer vilka kan tyckas haft en kraftig utveckling det senaste decenniet.

1.1 Bakgrund

Mjukvara är under ständig utvecklingen och i flera fall kan kostnaden för vidare underhåll och utveckling utgöra 40 % av den totala kostnaden för den utvecklade mjukvaran, denna siffra kan dock variera från fall till fall (Araújo et al., 2011). Samtidigt som underhåll är viktigt menar Seacord et al. (2003) att dessa kostnader och åtgång på resurser kan äventyra fortsatt utveckling av ny mjukvara. Seacord et al. benämner fenomenet *”legacy crisis”* och i en djupare mening menar de bland annat att de fortsatta stora kostnaderna gällande underhåll och vidare utveckling för befintlig mjukvara riskerar investeringar i utvecklingen av ny mjukvara. Konsekvensen av detta menar Secard et al. är att det finns risker att hamna i en *”Middle Ages of the information age”*, med avsaknad av moderna system.

Det finns i enlighet med Secard et al. alltså en risk att utvecklingen av ny mjukvara blir lidande då resurser går till att underhålla gammal mjukvara och inte till utveckling av ny mjukvara. Däremot menar Mens & Tourwé (2004) att fortsatta kostnader efter färdigutveckling är oundviklig. De menar att befintliga system fortsätter att utvecklas och modifieras för att möta nya krav, behov och syften, detta då processer och arbetssätt hela tiden utvecklas och den stödjande mjukvaran måste stödja dessa nya arbetssätt och processer. Detta får dels den konsekvens som nämns ovan, att en stor del av kostnaden för mjukvaran är knuten till underhåll men även att kodens komplexitet ökar då den tenderar att förändras på sådant sätt att den inte behåller planerad struktur och design.

Mens & Tourwé (2004) anser att detta inte går att lösa med att ändra de metoder som används vid mjukvaruutveckling (exempelvis ”Scrum”, ”spiral model” och ”XP”) då det är omöjligt att förutse hur kommande förändringar kommer ta sin form. Istället för att försöka förutse kommande förändringar anser Mens & Tourwé med flera att *”refactoring”* är den metod som bör anammas under det fortsatta underhållet av mjukvaran i syfte att upprätthålla lämplig strutkur i koden.

Som diskuteras ovan finns det litteratur som menar att underhåll och ”refactoring” tar upp mycket tid, så mycket tid att det kan påverka den vidare utvecklingen av ny mjukvara. För att på ett effektivare sätt utveckla och underhålla redan befintlig mjukvara har olika typer av ramverk utvecklats. Riehle et al. (1998) menar att ramverk har en sådan påverkan på ett mjukvaruprojektets resultat att istället som tidigare, haft fokus på vilket språk som skulle användas vid en viss typ av applikationsutveckling, finns det idag ett större fokus på vilket ramverk som är mest lämpligt för respektive applikation. Detta då ramverket gör utvecklingen och vidare underhåll effektivare och enklare. Winsett (2010) vill till och med beskriva utvecklingen av användningen av ramverk som en ”boom”:

> *”The past several years have marked a significant ’framework boom’, and almost everyone involved in web application development these days is a part of a new generation of ‘framework boomers’ ”* - Winsett (2010, s7)

Som uppsatsens rubrik antyder syftar begreppet ramverk i denna uppsats till ramverk för webbapplikationer. Dessa ramverk syftar till att stödja utvecklingen och underhållet av dynamiska *webbplatser, webbapplikationer och webbtjänster* [1].

[1] Andra begrep alternativt förkortningar för dessa är SaaS, SAP och molntjänster.

1.2 Problemdiskussion

Då ramverk för webbapplikationer [2] inte funnits en längre tid (exempelvis inom programmeringsspråket PHP kom de första ramverken först 2005) bör det finnas webbapplikationer som inte är baserade på något ramverk. Som belyses i bakgrunden menar Winsett (2010) med flera att det kan tyckas lämpligt att använda sig av ramverk vid utveckling av applikationer för att underlätta bland annat underhåll.

Utifrån detta kan det tyckas intressanta att utreda om befintliga applikationer som inte är baserade på ett ramverk skulle löna sig att överföra till ett ramverk, detta ur ett tidsperspektiv. Det kan dessutom vara intressant att utreda vilka komplikationer som eventuellt kan uppstå vid denna överföring. Applikationer som inte är baserade på något ramverk benämns vidare i denna uppsats som applikationer baserade på *fristående kod*. Med fristående kod menas programkod som har skrivits utan stöd av ett ramverk eller liknande

För att se om en sådan överföring är till fördel kan det även vara intressant att genomföra underhåll och uppdateringar i applikationen som fristående kod samt som överförd till ett ramverk.

Som diskuteras i bakgrunden finns det till exempel metoder för ”refactoring”, dessa metoder är dock inte avsedda för överföringar av applikationer från fristående kod till ett befintligt ramverk. I denna studie kommer därför en ny metod presenteras: ”toFramework refactoring”. ”toFramework refactoring” är den metod som används vid överföringen av applikationen från fristående kod till ett ramverk.

1.3 Problemformulering

Uppsatsen ämnar besvara följande två frågor:

1. Vilka komplikationer finns då en applikation baserad på fristående kod ska överföras till ett existerande ramverk?
2. Är det förmånligt ur tidsaspekt och underhållsaspekt, att överge en redan fungerande applikation skriven i fristående kod för att istället skapa samma funktionalitet i ett ramverk?

1.4 Syfte

Syftet med uppsatsen att belysa eventuella komplikationer som kan komma att uppstå när en webbapplikation där befintlig mjukvara kan anses rimlig att överföra till passande ramverk. Syftet är även att kunna visa om vidare utveckling och underhåll blir effektivare i valt ramverk, detta i förhållande till överföringen. Dessa resultat kan vara vägledande för de som underhåller applikationer skrivna i fristående kod.

[2] Till exempelvis ASP.NET, Yii, CakePHP och Play!

[3] Asynchronous JavaScript and XML är en teknik som används för att öka interaktiviteten för sidor på WWW.

[4] jQuery, Javscript bibliotek vars syfte är att underlätta klientskript för exempelvis animeringar och färdiga GUI:n.

1.5 Disposition

Uppsatsen består av sju kapitel varav det första var den ovanstående inledningen. Nedan presenteras de följande sex kapitlen i syfte att ge en enkel överblick av uppsatsens.

Inledningsvis kommer den teoretiska referensramen redogöra för vad som menas med ramverk och webbapplikationsramverk för denna uppsats. Följt av detta kommer begreppen ”refactoring” och underhåll förklaras, detta i syfte för att visa var inspirationen till begreppet och metoden ”to-Framework refactoring” har sin grund.

Metodkapitlet syftar till att tydliggöra hur uppsatsens studie är genomförd. I detta kapitel beskrivs exempelvis hur ”toFramework refactoring” har utvecklats och används . Metoden har bland annat inspirerats av ”refactoring”, ”SCRUM” och ”Singel Xtreme Programing” (SXP).

Följt av metodkapitlet kommer resultatkapitlet vilket är uppdelat i två delar. I den första delen kommer överföringen av applikationen redogöras, detta tillsammans med en kort reflektion över hur lång tid varje överföring tog i anspråk. Följt av detta kommer tidsdifferenser vid underhåll och uppdateringar redogöras mellan överförd applikation och applikationen som fristående kod.

I det näst sista kapitlet, analysen, kommer resultatet diskuteras i ljuset av uppsatsens syfte och frågeställningen. Analysen syftar till att ge ett bra underlag till slutsatsen. I det sista kapitlet, slutsats, presenteras de slutsatser som dragits utifrån resultatet och diskussionen. Denna avslutas med förslag på vidare forskning.

2 Teoretisk referensram

Inledningsvis kommer detta avsnitt fokusera på att ge en förtydligande bild av vad som menas med ramverk för webbapplikationer i denna uppsats. Detta följs av förklaring av begreppen underhåll och ”Refactoring” vilka ligger till grund för begreppet ”toFramework refactoring”.

2.1 Ramverk för webbapplikationer

Riehle (2000) menar att ett ramverk är en universal, återanvändbar plattform som används för att utveckla applikationer och program. Ramverk innefattar ofta kodbibliotek, api:er, stödprogram etc. Helt enkelt är ett ramverk olika komponenter som har sammanfogats för att passa vissa typer av utvecklingsprojekt. För denna uppsats syftar begreppet ramverk till engelskans *”web application frameworks”*. ”Web applikation frameworks” har i syfte att förbättra effektiviteten i skapandet av nya webbapplikationer.

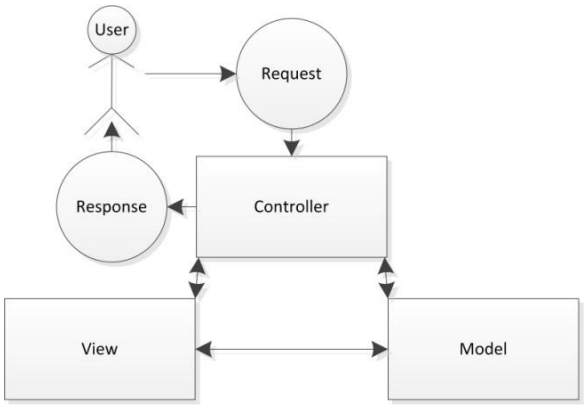
Dessa ramverk tillhandahåller oftast huvud funktionaliteter så som användarhantering, kopplingar gentemot databaser eller motsvarande samt uppsatta utvecklingsmiljöer. Ramverk kan förbättra utvecklarnas produktivitet och förbättra applikationens kvalitet, tillförlitlighet och säkerhet. Utvecklingen blir effektivare genom att utvecklarna kan fokusera på de unika krav som ställs på applikationen i stället för att spendera tid på att bygga grundläggande infrastruktur och designmönster (Winsett, 2010). Ett designmönster som ofta används är för dessa ramverk är *MVC*.

2.1.1 Designmönstret MVC

I ramverk avsedda för webbapplikationer används ofta designmönstret MVC som bygger på tre olika lager, *”model”, ”view”* och *”controller”*. Dess relation beskrivs nedan men gestalts även i figur 1. ”Model” som är ett annat namn för *domänlagret* är den del av programmet som bearbetar rådata och gör denna meningsfull. Rådata kan utgöras av data från eventuell databas, inmatning från användaren eller andra inbyggda funktioner såsom datumfunktioner. Då domänlagret endast ska hantera rådata hanterar den inte kopplingen till omvärlden, detta gör ”controllern”. (Reenskaug, 2003)

”Controllern” hanterar förfrågan från omvärlden, detta kan exempelvis ske genom en förfrågan från en klient till webbservern, ”controllern” i detta fall hanterar förfrågan och använder olika ”models” för att ta fram lämplig data för att presentera dessa data i en lämplig ”view” beroende på klientens förfrågan.

”Controllern” använder alltså ”view” filerna för att presentera den data som erhålls av domänlagret. ”View” filerna består till mesta dels av *HTML* men med vissa inslag av serverspråk för att presentera den data som erhålls av ”controllern” som i sin tur får datan från domänlagret. (ibid.)



Figur 1. Designmönstret MVC

2.1.2 Specifika tekniska fördelar

Förutom grundläggande infrastruktur och designmönster tillhandahåller moderna ramverk funktioner och plugins. Detta inte bara inom serversidan utan även inom CSS, HTML och JavaScript. Exempel på dessa egenskaper är AJAX[3], jQuery[4], ramverk för testning, funktioner för ”caching”, ”form-validation” med mera. Dessa egenskaper kan anses specifika för ramverk avsedda för webbapplikationer.

2.2 Definition av begreppet underhåll

När mjukvaran är implementerad påbörjas ”maintenance”, eller underhållsfasen. Underhåll av mjukvara kan beskrivas likt en process där mjukvaran fortsätter att anpassas till verksamhetens värdeskapande aktiviteter samt att korrigera fel (D'Hondt et al., 2010)

Som nämns ovan ställs nya krav från användarna och verksamheten, men krav kan även komma från andra håll. Den externa omgivningen utanför organisationen kan även den ha en påverkan som leder till nya krav, exempelvis kan ny lagstiftning eller ny teknologi som erbjuder nya möjligheter påverka mjukvarans utveckling. D'Hondt et al. (2010) benämner dessa omständigheter för ”Software Evolution” där systemet kan ses som en del av dess omgivning. Under tiden som omgivningen förändras ställs det nya krav på mjukvaran som utvecklas parallellt med dess omgivning.

2.3 Förändring och underhåll av mjukvara - ”Refactoring”

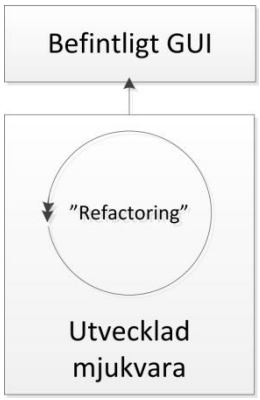
Enligt Veerraju et al. (2010) är ”refactoring” när kod omarbetas i syfte att göra den mer effektiv, enklare att underhålla, enklare att förstå samt enklare att arbeta med. Det innebär alltså inte att lägga till nya funktioner utan endast att förbättra de redan existerande funktionerna. En annan viktig aspekt vid ”refactoring” är att användargränssnittet inte ska påverkas av de förändringar som görs i koden, användaren ska alltså inte kunna märka att koden har förändrats, detta gestaltas i figur 2. ”refactoring” kan definieras enligt följande punkter:

- ”refactoring” utgörs av flera olika tekniker som syftar till att identifiera undermåliga designmönster i koden och förbättra dessa, utan att påverka mjukvarans synliga beteende.
- ”refactoring” syftar till att göra objektorienterad kod mer flexibel och återanvändbar.
- ”refactoring” har ett mål, att få varje enskild funktion att endast finnas på en plats i mjukvaran.

Som diskuterats tidigare finns det flera anledningar till varför det finns anledning till att fortsätta att förändra redan skriven kod, bland annat av andra krav på vad mjukvaran ska utföra. Veerraju et al. (2010) menar att programmerar under utvecklingen skriver kod med ”bad code smell” för att de vet att koden ändå kommer skrivas om i senare skede samt krav på snabb leverans. Som tidigare diskuterades i bakgrunden är alltså ”refactoring” något som kan anses nödvändigt och inte något som helt går att undvika.

[3] Asynchronous JavaScript and XML är en teknik som används för att öka interaktiviteten för sidor på WWW.

[4] jQuery, Javscript bibliotek vars syfte är att underlätta klientskript för exempelvis animeringar och färdiga GUI:n.



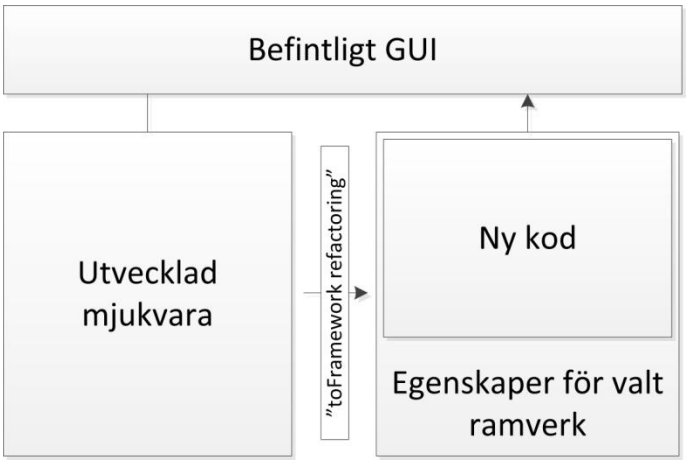
Figur 2. Modell över ”refactoring”

2.4 toFramework refactoring

Som diskuteras i problemdiskussionen finns det för de ovan diskuterade, ”refactoring” flera olika metoder. Dessa metoder beskrivs av flera författare för hur dessa processer ska gå till, men de utgör inte en metod för att gå över till ett ramverk vilket är aktuellt för denna uppsats.

Då ”refactoring” inte passar in som lämpligt tillvägagångssätt för uppsatsens studie, har det ansetts lämpligt att utveckla en ny metod; ”toFramework refactoring”. Till skillnad från ”refactoring” fortsätter alltså inte utvecklingen i den befintliga mjukvaran utan samma funktionalitet överförs till ett lämpligt ramverk utan att påverka dess GUI, detta gestaltas i figur 3.

”toFramework refactoring” kan på ett sätt verka mer komplext än ”refactoring” då den mjukvaran ska anpassas till valt ramverk samt behålla befintligt GUI. Som diskuteras i bakgrunden och problemdiskussionen bör det finnas en metod och ett arbetssätt för detta, på samma sätt som det finns för ”refactoring”. Denna metod kommer att utvecklas vidare i kommande kapitel.



Figur 3. Modell över ”toFramework refactoring”

3 Metod

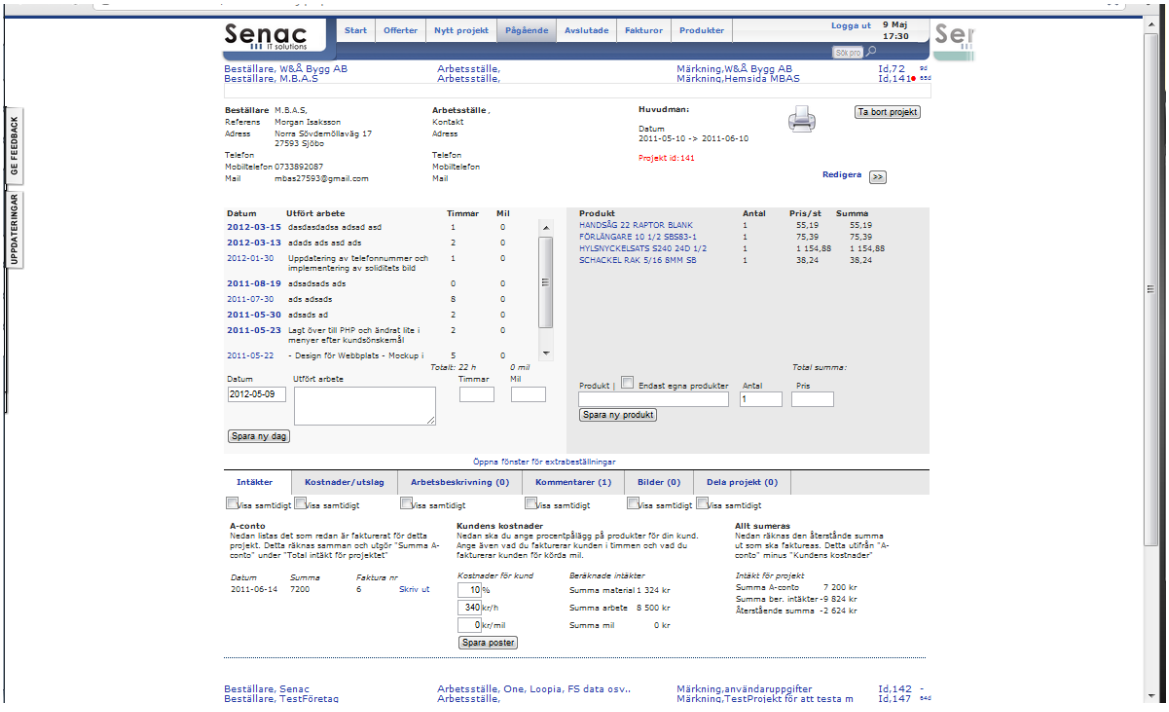
Inledningsvis beskrivs den aktuella applikationen som ska överföras samt valt ramverk, detta följs av hur metoden ”toFramework refactoring” har arbetats fram. Metoden har inspirerats av andra tidigare i litteraturen diskuterade metoder för dels ”refactoring” men även metoder för nyutveckling såsom ”Scrum”.

3.1 Tydliggörande av aktuell applikation samt ramverk

3.1.1 Applikation

”Struktur” är en applikation som hjälper hantverkare med att administrera projekt gentemot sina kunder, GUI visas i figur 4:a. ”Struktur” kan benämnas som en molntjänst av typen ”Software as a service” (SaaS). Enkelt uttryckt innehåller Struktur följande saker:

- Möjlighet att skapa offerter och koppla dessa till projekt
- Hantera pågående, avslutade och arkiverade projekt
- Innehåller tidsrapporteringen
- Skapar ordning bland projekt och möjlighet till uppföljning
- Enkelt hantera produkter och knyta dessa till projekt
- Skapar fakturor utifrån projekt med två klick



Figur 4. GUI:t för mjukvaran som ska överföras

3.1.2 Ramverket - Yii

Ramverket som applikationen har överförts till heter ”Yii” [5]. Yii är ett *“open-source”* ramverk som började utvecklas 2008. Ramverket är skrivet i PHP. Detta är ett av många ramverk men ett av de modernaste inom programspråket PHP.

[5] Ramverket Yii:s webbplats: http://www.yiiframework.com

3.2 Litteraturstudier för att kunna skapa metoden ”toFramework refactoring”

För att kunna göra en överföring av ovan redovisade applikation till ovan redovisade ramverk gjordes en enkel litteraturstudie för att hitta lämpliga metoder och modeller för att kunna skapa metoden ”toFramework refactoring”. Metoden ”toFramework refactoring” kommer redovisas i följande kapitel.

I enlighet med Forsberg et al. (2008) är denna uppsats bakgrund baserad på en enkel litteraturstudie. Syftet med en enkel litteraturstudie i det inledande skedet är att motivera och underbygga uppsatsens syfte och bakgrund.

För att på ett systematiskt sätt hitta lämpliga studier relaterade till uppsatsens ämne, definierades flera sökord. Detta i enlighet med Hearn et al. (1999) . Sökorden är delvis inspirerade av de ”key words” som fanns i de studier som hittades under den tidigare nämnda förstudien. Förutom inspiration från ovan nämnda ”key words” har även vissa sökord definierats utifrån denna uppsats bakgrund.

Följande sökord användes: ”Refactoring”, ”Restructuring”, ”Software framework”, ”Software framework refactoring”, ”web application framework”, ”Software framework development”, ”Agile framework development”, ”Software development”.

Efter att sökorden var definierade användes i huvudsak sökmotorerna google (google.se), Google scholar (scholar.google.se) samt summon supersök (http://gothenburg.summon.serialssolutions .com). Förutom sökningar i ovan nämnda databaser gjordes även en genomgång av relevant litteratur i förhållande till tidigare nämnda sökord vid Göteborgs Universitetsbibliotek.

Dessa sökningar resulterade i kontakt med andra begrepp och lämpliga sökord som även de utforskades. Däremot resulterade oftast dessa sökningar i olika sidospår vilket lätt riskerade att ämnet gleds ifrån, därför var det bra att kunna falla tillbaka på de fördefinierade sökord som listats ovan.

Vid sökningen av tidigare studier sparades de som vid första anblick ansågs relevanta ned, eller vid fallet av böcker lånades de hem. Efter att ett stort antal artiklar och böcker hade samlats in påbörjades det en grov sortering av dessa. Sorteringen skedde efter en enkel innehållsanalys som baserades på förekomsten av tidigare nämnda sökord samt dess relevans i förhållande till denna uppsats syfte och problemformulering.

Då vi lever i ett samhälle där det blir allt lättare att sprida och dela information i olika medier finns det, som Andersen och Schwencke uttrycker det: ”*goda motiv att vara källkritisk”* (Andersen, Schwencke, 2009:92).

Vid internetbaserad databassökning är det viktigt att komma ihåg att det finns många aktörer som vill påverka opinionen på olika sätt och kan därför vinkla olika data utifrån önskat ändamål. Här är det viktigt att vara kritisk och försöka hitta ursprungskällan till information då den kan vara mer nyanserad och även undersöka om den är trovärdig.

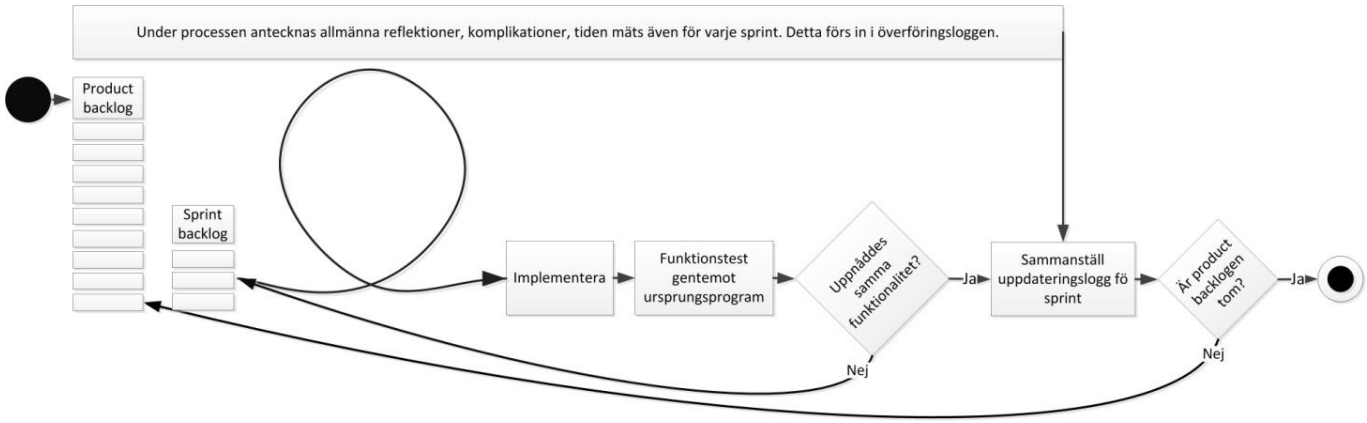
3.3 Metoden ”toFramework refactoring”

Mens & Tourwé (2004) menar att ”refactoring” processen består av ett antal huvudsakliga aktiviteter. Utifrån deras föreslagna aktiviteter har nedanstående aktiviteter arbetats fram i syfte att strukturera upp det övergripande arbetet med ”toFramework refactoring”.

1. Bestämma vad och på vilket sätt mjukvaran ska omarbetas.
2. Utarbeta metoder för att säkerställa att samma funktionalitet finns kvar efter omarbetningen.

3. Utföra omarbetningen.
4. Testa att tidigare funktionalitet kvarstår.

Dessa aktiviteter är mer övergripande och skapar de grundläggande ramarna för arbetet med metoden ”toFramework refactoring”. För att få en överskådlig förståelse för arbetet med ”toFramework refactoring” har en modell framarbetats, se figur 5. I följande kapitel kommer punkterna ett till fyra som nämnts ovan förklaras ytterligare i relation till metoden.



Figur 5. Modell över metod för ”toFramework refactoring”

3.3.1 Vad och på vilket sätt ska mjukvaran omarbetas

Innan ”toFramework refactoring” påbörjas var det som Mens & Tourwé (2004) menar viktigt att bestämma vad som skulle omarbetas och hur detta skulle göras. För att på ett bra sätt konkretisera vad som skulle utföras upprättades en *”product backlog”*. ”Product backlogen” härstammar från systemutveckling metodiken ”Scrum” som utvecklades av Jeff Sutherland och Ken Schwaber i början av nittiotalet (Scwaber, 1999). Fler delar av ”Scrum” har vidare även används vilka kommer redogöras utförligare nedan.

Vilka paket ”product backlogen” skulle utgöras av definierades utifrån vad som var lämpligt för den aktuella applikationens egenskaper samt vad som var rimligt att utföra samtidigt utifrån ramverkets egenskaper. Dessa paket har i enlighet med Schwaber & Sutherland (2010) sedan tilldelats olika mått av prioritering. Denna prioritering resulterade även i en prioriteringsordning som paketen lämpligtvis skulle utföras i (ibid.). Dessa prioriterings parametrar var som följer

- Bedömd implementerings tid
- Bedömd komplexitet
- Bedömd implementeringsrisk (kunde funktionen inte vara möjlig att överföra fick den hög prioritering)

Gällande frågan om hur omarbetningen ska ske, har det på förekommen anledning gjorts efter valts ramverk (Yii) syntax och metodik. Dess arbetsmetoder menar utvecklarna av ramverket vara en form av *”best practice”*[6] för hur webbapplikationer bör konstrueras.

[6] Best Practice, en teknik, metod eller motsvarande som anses mer effektiv att generera eller leverera ett önskat resultat än någon alternativ teknik eller metod.

3.3.2 Metod för test av funktionalitet efter överföring

Detta steg beskriver ”Funktionstest gentemot ursprungsprogram” i figur 5 ovan. Definitionen är att den överförda delen ska uppnå samma funktionalitet som existerande applikation uppnår idag. Applikationen testades utifrån två vedertagna principer, *”black box testing”* och *”white box testing”* vilka beskrivs nedan, detta efter Gustafsson (2002).

För ”Black box testing” utgår testpersonen utifrån givna input och kontrollerar att rätt output genereras i förhållande till inputen. ”Black box testing” kan ses som en svart låda där testpersonen för in input och får ut output och analyserar inte vad som ligger bakom hanteringen av datan. Dessa tester har utförts på enklaste sätt av hantverkare som redan använder applikationen. Testerna har genomförts vid flertalet tillfällen när någon ny del av applikationen var färdigöverförd.

En annan metod som Gustafsson (2002) diskuterar är ”white box testing”, vid ”white box testing” kan mjukvaran ses som en transparent låda där det går att se hur hela omvandlingen från input till output sker. För ”white box-testing” följs de olika informationsflödena genom systemet för att se att olika typer av input hanteras på rätt sätt så att rätt output ges.

Gällande ”white-box testing” för denna applikation har ramverkets inbyggda testfunktioner som byggs parallellt med utvecklingen av applikationen används. Detta har gjorts samtidigt som mer traditionell ”white-box testing” utförs, genom visuell kontroll.

3.3.3 Metod för överföring av applikation

Detta steg symboliseras av cirkeln med en pil i figur 5 ovan. Tidigare har vilken och på vilket sätt applikationen överförts diskuterats, även hur den har testats har beskrivits. Detta leder oss fram till vilket sätt den faktiska överföringen har skett

I enlighet med Schwaber & Sutherland (2010) användes samma metod som för ”Scrum” av hanteringen av ”product packlog” och ”*sprint backlog*”, det som överförts från ”product backlog” till ”sprint backlog” utfördes under följande sprint.

Då arbetet med denna uppsats har utförts av en person ansågs det inte lämpligt att utgå ifrån ”Scrum”:s regler för sprinten eller vidare delar av ”Scrum”. För metod under sprinten har inspiration hämtats från Agarwal & Umphress (2008) som utvecklat en metod för hur en person bör driva ett mjukvaruutvecklingsprojekt med bakgrund från ”Xtreme Programming”. Beedle & Schwaber (2002) hävdar att ”Scrum” och XP kan med fördel kombineras. Utifrån Agarwal & Umphress utarbetade metod användes följande metod under vardera överföringssprint.

1. Krav - Klargöra uppnådd funktionalitet efter sprint
2. Bryt ner - Brutit ner aktuellt paket i minde moment
3. Sortera - Sorterat momenten i lämplig ordning
4. Koda - Kodat nya delar enligt sorterad ordning
5. Visuell kontroll - Visuellt kontrollerat koden
6. Implementera överföringen
7. Test - Testa gentemot krav (”white box testing” och ”black box testing”)
8. Avslut eller ny iteration – Klarades inte testen, genomför en ny iteration

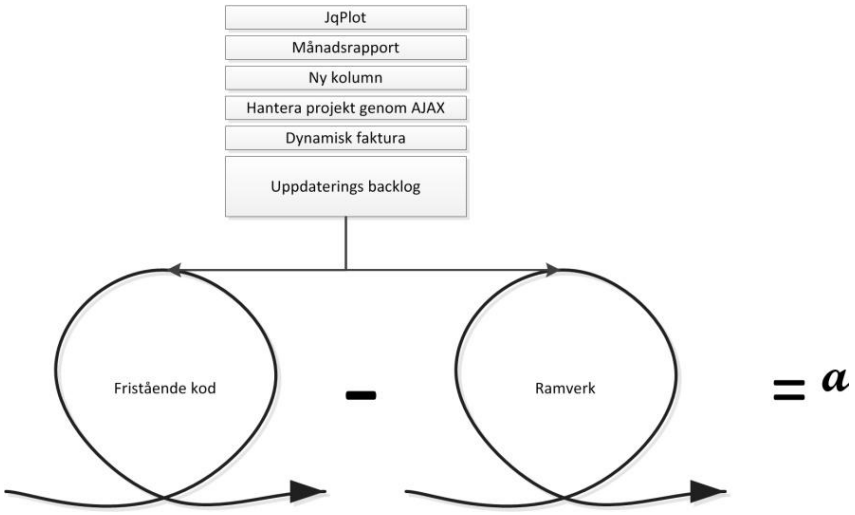
Sprinten utfördes alltså inte som en traditionell ”Scrum” sprint. Under tiden att överföringen genomfördes loggades även aktiviteter och komplikationer av intresse i överföringsloggen, detta gestaltas överst i figur 5. I överföringsloggen loggades även tiden för den tid sprinten tog att genomföra. Syftet med loggen var att skapa underlag som kunde ge svar på uppsatsens problemformulering angående komplikationer och tidsaspekter.

3.4 Metod för utvärdering av eventuella tidsvinster

När mjukvaran var överförd i ramverket, påbörjades arbetet med att svara på uppsatsens andra problemformulering:

Är det förmånligt ur tidsaspekt och underhållsaspekt, att överge en redan fungerande mjukvara för att istället skapa samma funktionalitet i ett redan färdigutvecklat ramverk?

För att undersöka hur uppdateringar och vidare underhåll skilde sig i tid från applikationen som fristående kod och applikationen som ramverk gjordes följande tester, enligt figur 6. Testet bestod av att samma uppdateringar i både i applikationen i ramverket samt applikationen som fristående kod utfördes. För att resultat skulle bli korrekt och att det inte skulle kunna gå att dra nytta av att uppdateringen gjorts i den ena applikationen, har två olika personer utfört uppdateringarna i applikationen utvecklad på fristående kod respektive applikationen överförd till ramverket. Personerna har haft samma förutsättningar då de har haft samma kunskap om applikation och programmering.



Figur 6. Tidsjämförelse mellan ramverk och friståendekod – underhåll och uppdateringar

$$\sum_{k=1}^{n} a_k = \text{Resultat för uppdatering}$$

Figur 7. Ekvation för summering av tidsdifferenser

Efter att genomfört dessa uppdateringar samlades flera tidsdifferenser in vilka summerades enligt figur 7. Därefter ställdes summan i förhållande till den totala tiden (sammanställd från överföringsloggen) för överföringen av applikationen in i ramverket. Resultatet från denna uträkning gav en indikation om det var ur en tidsaspekt lämpligt att övergå till ett ramverk. Detta redogörs i det kommande resultat kapitlet.

4 Resultat

I detta kapitel presenteras delar av överföringsloggen, detta då överföringsloggen i sin helhet i förhållande till sin storlek inte skulle tillföra uppsatsen någon ytterligare dimension. Överföringsloggen följs av resultatet av de genomförda uppdateringarna. Till vardera del förs även resonemang gällande eventuella komplikationer och reflektioner.

4.1 Steg ett - överföring av applikationen

Som nämns ovan har inte alla överföringar tagits med i överföringsloggen. I överföringsloggen nedan har de mest generella överföringarna tagits med då de till sin natur liknar många andra överföringar.

4.1.1 Överföringslogg

Överföringsloggen består av tre kolumner, vilka beskrivs nedan:

Kolumn ett:

- Paket (Namn, prioritering) – Vilket namn och prioritering paketet har haft i ”Product backlogen”.
- Allmänna reflektioner – Under denna rubrik beskrivs nämnvärda reflektioner som gjordes för respektive överföring.
- Eventuella komplikationer – Under denna rubrik beskrivs nämnvärda komplikationer för respektive överföring.

Kolumn två:

- Klarat test – Överföringen har klarat de kriterier för testning som sattes upp innan överföringen påbörjades.

Kolumn tre:

- Tidsåtgång – Tiden det tog att genomföra överföringen för aktuellt paket, denna tid inkluderar tiden för testen.

Paket (Namn, prioritering), allmänna reflektioner och eventuella komplikationer	**Klarat test**	**Tidsåtgång**
Inloggning, 1 **Allmänna reflektioner** Det som kan vara en webbapplikations akilleshäl är ofta dess problem med säkerhet, speciellt vid applikationens inloggning. Något som ofta diskuteras är risken för ”Sql injections”[7]. Det aktuella ramverket tillhandahåller en struktur som gör att utvecklaren inte behöver oroa sig för denna risk. Inloggningsfunktionen är speciellt utsatt då denna är åtkomlig för icke inloggade användare. Däremot bör det understyckas att, oavsett om applikationens inloggnings funktion är säker kan ”Sql injection” utföras mot andra öppna delar av applikation	Ja	1h

[7] ”Sql injections” är när indata i form av sql skickas gentemot en databas i syfte att få tillgång till data som användaren inte är berättigad till.

som kan leda till att sparade användaruppgifter kommer i fel händer, vilket undviks genom ramverkets struktur. **Eventuella komplikationer** Att skapa en inloggning för användare är något som autogenereras när en ny applikation skapas i ramverket. Det vill säga att det inte finns några komplikationer för inloggningsfunktionen i sig, varken för att skapa eller att anpassa denna för aktuellt tillämpningsområde då inloggningsfunktioner till sin natur är väldigt lika.		
2. Användarnivåer **Allmänna reflektioner** Applikationen består av tre användarnivåer, vilka har olika behörigheter. Ramverket tillhandahåller även funktioner för detta. Förutom att uppdatera databasen med kolumnen ”level” var det endast några få rader som behövdes för att kunna hantera de olika användarnas användarnivåer. Då ramverket även tillhandahåller strukturen för ”Role-based access control” som är en vedertagen metod för hantering av användarrättigheter (Ferraiolo & Kuhn, 2007) skapar detta goda möjligheter för fortsatt utvecklingen av applikationen, vilket inte hade varit fallet i den fristående koden. **Eventuella komplikationer** Inga komplikationer.	Ja	2h 30min
3. Skapa CRUD[8] för projekt **Allmänna reflektioner** Metodiken för att skapa och hantera olika datamodeller för denna applikation är egentligen den samma, CRUD. Som nämnts tidigare har vissa paket valts ut för att presenteras i resultatet då de representerar flera liknande överföringar. Tillvägagångssättet för att skapa ett projekt liknar till del utsträckning hur kontakter, företag och användare skapas. Att skapa en struktur för CRUD för ovan nämnda objekt går i princip i ett musklick. Utifrån de redan färdiga databastabellerna skapas modeller, ”controllers” samt ”view”-filer vilket gör det möjligt att direkt genomföra CRUD genom genererat GUI (view-filerna). Det som tar tid i detta är anpassningen till aktuellt GUI för applikationen som ska överföras samt viss konfiguration av hur datan ska presenteras och hanteras. **Eventuella komplikationer** Då strukturen för CRUD skapas automatisk för vardera modell finns det inte så många komplikationer att diskutera. Det som finns att lyfta gällande komplikationer är de som undviks genom att slippa att skriva allt från scratch då det finns stor risk att man fastnar vid grundläggande CRUD funktioner. Ramverk gör det möjligt att snabbt komma igång.	Ja	3h
4. Skapa/hantera dagar	Ja	6h

[8] CRUD, Create, Read, Update och Delete vilka är de fyra grundläggande funktionerna för hantering av persistent data.

Allmänna reflektioner Som nämns ovan skapas CRUD-funktionerna väldigt snabbt och enkelt. Nästa steg i detta är att sammanfoga projekt med dagar då ett projekt kan ha flera dagar. En dag kan även ha olika karaktärsdrag. Skapa/hantera dagar påminner bland annat mycket till sin natur som att spara/hantera produkter vilka även de tillhör ett specifikt projekt. Då projekt kan ha flera dagar har de en relation. Relationer sätts enkelt i ramverket och autogenereras även om de är angivna i de tabeller som modellerna skapas utefter. Detta gör att det enkelt går att nå de relaterade dagar för respektive projekt utan att skriva omfattande funktioner. För hanteringen av dagar används mycket AJAX, ramverket har även ett bra stöd för detta. Ett exempel för smidig användning av AJAX är vid uppdatering av data med ett formulär där det räcker att skriva ”AjaxButtonSubmit” istället för ”submitButton” för att informationen ska sparas via AJAX. **Eventuella komplikationer** Som diskuteras ovan innefattar hanteringen av dagar en del AJAX. Problematiken vid utveckling av olika AJAX-funktioner är att ramverket inte tillhandhåller någon felsökningsfunktion. Detta göra att det vid vissa tillfällen blir svårt att lokalisera vad som är fel om de eventuellt inte skulle fungera som önskat. Skulle funktionerna istället gjorts i fristående kod hade det funnits bättre möjligheter att prova delar av de funktioner där felet eventuellt fanns och på ett mer metodiskt sätt lösa problemet.		
<u>9. Hantera projekt</u> **Allmänna reflektioner** Det har tidigare redogjorts för hantering av dagar som är relaterade till projekt. Detta är en del i hanteringen av projekt men ett projekt består av många fler attribut och relationer vilket skapar flera funktioner och rader av kod. Då ramverket möjliggör uppbyggnad efter DRY[9] så blir koden väldigt kort och enkel att förstå även då det finns flera relationer, affärslogik och attribut för i detta fall objektet projekt. Till skillnad från tidigare redovisade överföringarna innehåller denna överföring mer affärslogik, det vill säga specifik hantering av viss data. Detta har inte inneburet komplikationer men då den fristående koden inte gick att återanvända behövdes helt ny kod skrivas. Däremot var det till stor fördel att kunna utgå ifrån den redan skapade affärslogiken. **Eventuella komplikationer** Något som uppmärksammats under arbetet med överföringen är att det är svårt att ta ”genvägar” i ramverket, detta är i sin sak bra men blir till nackdel då det i vissa lägen kan vara till fördel att testa enkla funktioner för att undersöka om det är rätt väg att gå ur ett större perspektiv. Detta blev speciellt tydligt under denna överföring då det var flera funktioner som skulle fungera tillsammans.	Ja	30h

[9] DRY - Dont Repeat Yourself, princip vid mjukvaruutveckling vars ide bygger på att funktionalitet inte ska upprepas.

20. Skapa/hantera mall för utskrift **Allmänna reflektioner** Även för utskrift finns det stöttning i ramverket. Ramverket tillhandahåller CSS[10]-filer som är avsedda för just utskrifter. Fördelen med detta är att de kan finnas olika grundinställningar i olika webbläsare som kan göra att utskrifterna i olika webbläsare ser olika ut, detta undviks då ramverket tillhandahåller CSS som täcker de vanligaste webbläsarna. Den andra delen i detta var att skapa funktioner för utskrifter som skulle kunna återanvändas vid utskrifter oavsett vad som skulle skrivas ut. Detta tog i princip samma tid att skapa som det skulle gjort i fristående kod, fördelen var de kompletta CSS-filerna som man kunde luta sig tillbaka på. **Eventuella komplikationer** Inga komplikationer upplevdes under denna överföring.	Ja	5h
16. Spara/hantera bilder **Allmänna reflektioner** Som väntat fanns det inga för eller nackdel gällande hanteringen av bilder i ramverket. De ”plugins” som finns för bildhantering, till exempel ”ImageMagick”[11] funkade utan större problem att kombinera med ramverkets struktur. **Eventuella komplikationer** Inga komplikationer upplevdes under denna överföring.	Ja	4h
22. API gentemot mobilapp genom JSON **Allmänna reflektioner** Att skapa ett API i JSON[12] var inte avancerat och stöds även detta av ramverket. Mycket av den fristående koden gick även att återanvända, vilket inte varit fallet för de tidigare överföringarna. Gällande auktorisering gick det bra att använda den redan framarbetade inloggningsfunktionen för att tillhandahålla rätt data för respektive användare. **Eventuella komplikationer** Inga komplikationer upplevdes under denna överföring.	Ja	3h
	Summa tid	54h 30min

Tabell 1. Delar av överföringsloggen

[10] Cascading Style Sheets, stilmallar för layout av främst HTML-dokument.

[11] Programtillägg som installeras på aktuell webbserver som kan hantera bilder.

[12] JavaScript Object Notation, kompakt textbaserat format som är avsett för utbyte av data mellan eller inom applikationer.

4.1.2 Sammanfattning av allmänna komplikationer samt reflektioner vid överföring

Som går att utläsa av överföringsloggen ovan har överföringen gått väldigt smidigt. Av de komplikationer som har uppstått (dock få räknade) har de i stort haft sin grund i oerfarenhet av mig som programmerare.

Ramverket har aldrig upplevts som begränsande när överföringar har genomförts, detta trots att ramverket bygger på mycket autogenerad kod och en grundläggande struktur så har denna struktur inte hindrat att skapa den struktur som har passat för den redan utvecklade applikationen.

Strukturen på databasen har inte heller den behövts förändras eller uppdateras för att passa ramverket. Ramverket har alltså inte ”ställt” krav på en viss databasstruktur och har genererat lämplig funktionalitet efter den aktuella databasen.

Något som var oväntat var att anpassningen till önskat GUI tog längre tid än väntat. Då det finns färdiga ”Widgets”[13] som gjorde att data snabbt och enkelt kunde presenteras tog det desto längre tid att konfigurera CSS-filerna för att skapa önskat GUI. Alternativet hade varit att skriva egna CSS-filer och implementera dessa men det hade inte varit förenligt med DRY principen.

4.1.3 Sammanfattande reflektioner för utslag av tid vid överföring

För de första överföringarna var det oklart om överföringen skulle gå att genomföra. Att överföringen kunde genomföras kunde först säkerställas efter cirka 40 timmar då de mest tekniskt avancerade paketen var överförda.

I utdraget från överföringsloggen ovan är det som bekant endast delar av överföringsloggen som presenterats, sammanlagd tid för dessa överföringar blev 54 timmar och 30 minuter. Sammanlagt har överföringar gjorts av en total tid av 100 timmar, detta innefattar dock inte alla överföringar som skulle behövas göras för att applikationen ska vara helt fungerande i ramverket. Uppskattningsvis skulle det ta ytterligare 60 timmar för att överföra de resterande delarna av applikationen.

Förutom den tid som lagts på överföring har även 10 timmar lagts på att installera ramverket. I stort bestod detta i att skaffa rätt versioner av behövlig mjukvara (apache och php), vilket var angivet på ramverkets webbplats. Det som även behövdes göras var att uppdatera serverns miljö- och systemvariabler.

Total tid för överföring blev uppskattningsvis 170 timmar där 10 timmar utgörs av installation av ramverket, 100 timmar av faktiskt genomförda överföringar samt 60 timmar för återstående överföringar.

Som kan utläsas från överföringsloggen sparades mycket tid då ramverket automatiskt skapade CRUD funktionerna utifrån den redan befintliga databasen. Det som är väsentligt i sammanhanget är att databasens utformning samt applikationens GUI har tagit cirka ett år att utveckla i samråd med de som använder applikationen. Detta diskuteras ytterligare i det kommande kapitlet 7, diskussion.

[13] ”Widget” är ett script som utifrån givna parametrar generar ett färdigt användargränssnitt för den data som önskas presenteras.

4.1.4 LOC och autogenererad kod

Som tidigare nämnts sparades mycket tid genom att kod autogeneras, till exempel funktioner för auktorisering, ”form-validiation”, CRUD med mera. Denna autogenererade kod blir av betydande massa och sparar mycket arbete. Detta kan ses som en ytterligare faktor som tidsmässigt gör det möjligt att överföra den redan fungerande applikationen till ett ramverk.

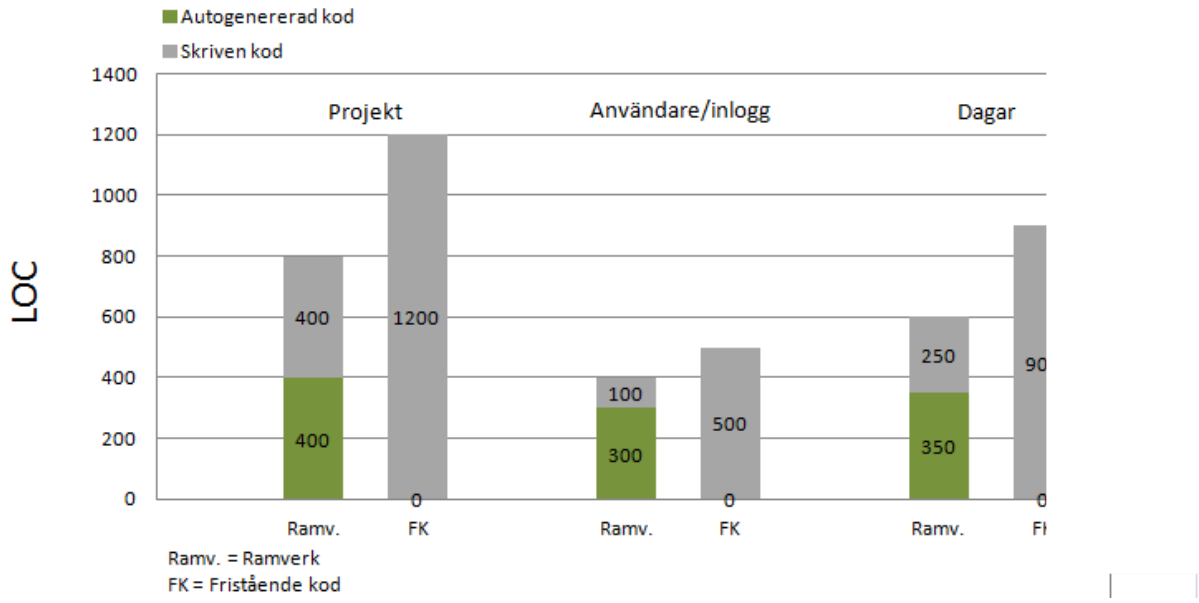
Gällande resultaten om mängden kod är det viktigt att skilja på grundläggande kod som skapas när en ny applikation genereras av ramverket och den kod som autogeneras för specifik funktion för applikationen.

Den grundläggande koden skapar den inbyggda funktionaliteten som är orsaken till den mindre kod som behövs i ”controller”, ”model” och ”view”, vilka är de filer man i huvudsak arbetar med. Det är således endast koden i ”controller”, ”model” och ”view” som summeras i figur 8.

Ett faktiskt mått på mängd kod är ”Lines of code” (LOC)[14]. LOC används ofta vid kostnadsbedömningar vid utveckling av mjukvara, exempel på metoder som bygger på LOC är COCOMO metoden. En intressant fråga är hur till exempelvis denna metod påverkas då mycket kod autogeneras. Med bakgrund från detta har LOC sammanställts för tre paket som har överförts, i figur 8 visas summan av LOC för respektive paket både i ramverket och som fristående kod.

De tre paket där en jämförelse har gjorts är projekt, användare/inlogg och dagar. För vardera paket visas LOC som krävs för paketets funktionalitet i ramverket respektive för samma funktionalitet skriven i fristående kod.

Detta är dock en svår uppskattning att göra då mycket funktionalitet för applikationen som fristående kod ligger i utskriftsfilerna men de angivna värdena anses ge en representativ bild över fördelningen av LOC.



Figur 8. Stapeldiagram över fördelning av autogenererad och skriven kod

[14] Lines Of Code, används ofta för att beskriva en mjukvaras omfattning och storlek och syftar till att räkna ett programs rader av kod.

4.2 Steg två - tidsdifferens vid underhåll

Efter att överföringen var gjord, började nästa steg i studien, att undersöka om den tid som det hade tagit att överföra applikationen till ett ramverk gick att tillgodogöra i det kommande underhållet av applikationen. För att mäta detta upprättades en uppdateringslogg i enlighet med vad som tidigare redogjorts.

4.2.1 Uppdateringslogg

Uppdatering	**Tid, fristående kod**	**Tid, ramverk**	**Differens**
Hantera projekt genom AJAX	2h 30min	30min	-2h
Skapa mer dynamisk hantering av fakturan	6h	4h	-2h
Lägga till kolumn i databasen samt att få denna lika tillgänglig i applikationen	1h	30min	-3h
Sammanställa data för månadsrapporter	2h 3min	2h	-0.5h
Implementering av JqPlot (externt JavaScriptplugin som ritar grafer)	2h	2h	0
		Summa tid	-7.5h

Tabell 2. Tidsdifferens för uppdateringar mellan fristående kod och ramverk

4.2.2 Sammanfattning av allmänna komplikationer samt reflektioner vid uppdateringar

Vid uppdateringarna var det tydligt att ramverket är mer lättarbetat än den fri stående koden. De främsta orsakerna till detta är:

- MVC strukturen underlättar - Att affärslogiken och övergripande kontrollfunktioner ligger separerat gör det enkelt att utföra uppdateringarna på rätt ställe.
- Mindre kod - Den mindre textmängden gör det betydligare enklare att skaffa sig en överblick över applikationen vilket gör att det går fortare att fastställa vad som ska ändras.
- Ramverkets inbyggda funktioner – Som tidigare diskuteras tillhandahåller ramverket flera färdiga funktioner, dessa var till hjälp vid överföringen men förenklade även arbetet med uppdateringarna i den mening att det sparade tid.

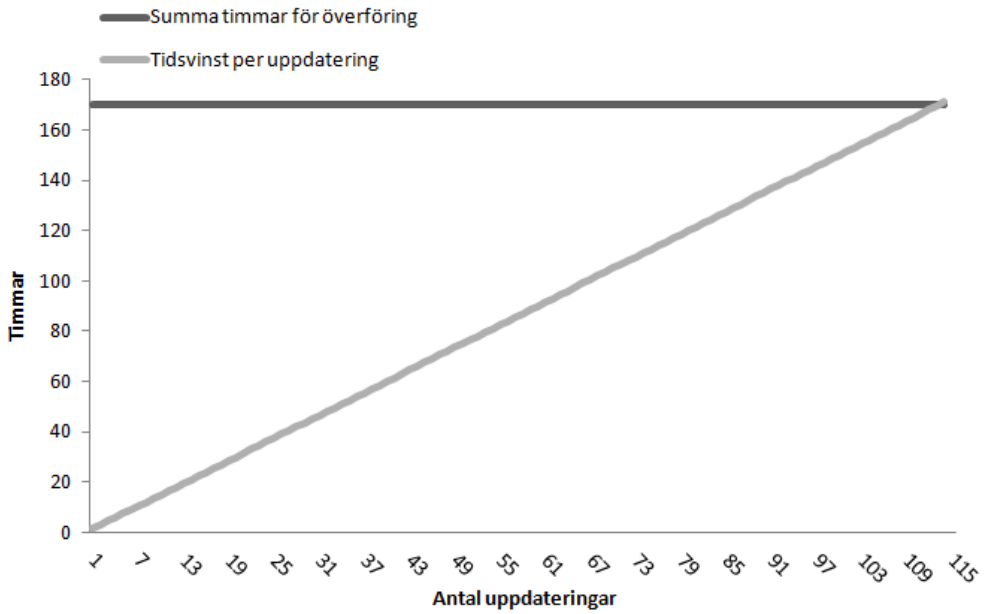
Inga nämnvärda komplikationer upplevdes, dessa var i så fall i den fristående koden.

4.2.3 Sammanfattande reflektioner för utslag av tid vid uppdateringar

Gällande uppdateringar var det ingen större skillnad i tid mellan ramverket och den fristående koden. De stora tidsvinster som fanns vid överföringen då mycket kod autogenererades fanns inte på samma sätt då ingen kod autogenerades vid uppdateringarna. Det bör påpekas att detta gäller de uppdateringar som redovisats ovan, det kan anses rimligt att vid andra uppdateringar utnyttja ramverkets funktion att autogenerera viss funktionalitet.

Ramverket är till viss del tidssparande då detta tillhandahåller mycket funktionalitet som annars skulle behövas byggas separat, men den största vinsten är minimeringen av mängden kod och den enklare överblicken av samtliga funktioner och dess relationer. Tid behövdes inte heller på samma sätt läggas på att testa och gå igenom tidigare skriven kod för att se om relationer och uppbyggnad var korrekt utfört. Gällande uppdateringarna kan de sammanfattas med att det var tidssparande men inte i den utsträckning som väntat.

Som det konstaterades tidigare skulle det uppskattningsvis ta cirka 170 timmar att överföra applikationen. En enkel genomsnitträkning av uppdateringarna visar att det tog cirka 1,5 timmar mindre tid att genomföra uppdateringen i ramverket i förhållande till den fristående koden. Utifrån detta krävs alltså drygt 110 uppdateringar för att tiden det tog att överföra applikationen till ramverket ska vara tillgodoräknad, detta illustreras i figur 9 nedan. Detta resultat diskuteras vidare i det kommande kapitlet.



Figur 9. Diagram över antalet uppdateringar i förhållande till tiden för överföringen

5 Diskussion

Diskussionen kommer inledas av en kortare utvärdering och diskussion om metoden ”toFramework refactoring”. Efter detta kommer överföringen analyseras och diskuteras under rubriken ”Tidsperspektivet” i enlighet med problemformuleringen. Avslutningsvis kommer mjukvaruutvecklingen ur en mer generell infallsvinkel diskuteras.

5.1 ”toFramework refactoring”, användbar?

För att genomföra överföringen på ett metodiskt sätt arbetades metoden ”toFramework refactoring” fram. Efter genomförd överföring är den sammanfattande bilden att metoden fyllde sitt syfte.

Med metoden ”toFramework refactoring” blev arbetet med överföringen strukturerat och således effektivt. Att använda ”Scrum” eller någon annan liknande metod hade också fungerat men med risken att missa steg som var viktiga för överföringen och studien i sig. Bland annat var överföringsloggen av stor vikt då denna skapade underlag för att i efterhand kunna analysera överföringen. Denna kan anses minst lika viktigt att använda sig av vid arbete med liknande överföringar utanför ramarna av uppsatsskrivande. Detta för att kunna göra uppföljning på överföringen gentemot uppsatta mål och i efterhand kunna kontrollera vad som är överfört respektive inte.

5.2 Tidsperspektivet

Från resultatet framgår det relativt tydligt att inga större komplikationer upplevdes under överföringen, därför lyser en diskussion om komplikationer i detta kapitel med sin frånvaro. I följande kapitel förs istället en diskussion om de anledningar som anses viktiga för en framgångsrik överföring.

Som tidigare diskuteras ligger de största fördelarna i att arbeta med ramverk i att mycket grundläggande funktionerna såsom CRUD, inloggningsfunktioner och formhantering autogenereras och struktureras upp automatiskt. Det är viktigt att påpeka i detta sammanhang att det bland annat är dessa autogenerade funktioner som gör det möjligt ur ett tidsperspektiv att göra överföringen, detta trots ramverkets andra fördelar.

En faktor som bör beaktas vid beslutet om en överföring ska genomföras eller ej är applikationens behov av vidareutveckling. Som påvisats i uppsatsens resultat krävs det drygt 110 uppdateringar för att överföringen ska vara till fördel ur ett tidsperspektiv. Det är bland annat med bakgrund till dessa siffror som en uppskattning bör göras om en överföring kan anses aktuell eller ej. Förutom tidsbesparning vid vidare utveckling skapar även ramverket en viss struktur vilket programmeraren får anpassa sig till för att dra störst nytta av ramverkets funktionalitet. Detta får även den fördel att flera programmerar bygger sina funktioner på liknande sätt vilket underlättar om flera programmerare arbetar på samma projekt vilket troligen också är tidssparande.

5.2.1 Mycket funktionalitet på kort tid – hur är det möjligt?

En komplett överföring av aktuell applikation skulle som tidigare påvisats ta cirka 170 timmar. Detta låga antalet timmar, i förhållande till vad applikationen har tagit att utveckla, kan ha sin grund i att den redan utvecklade applikationen har fungerat som ”*wireframes*” [15]. Förutom att applikationen har fungerat som ”wireframes” har även eventuell problematik gällande affärslogik genom denna redan varit känd, vilket har underlättat uppbyggnaden i ramverket. Att den redan

[15] ”Wireframes”, schematiska bilder över en applikations layout och navigationsmöjligheter

utvecklade applikationen var till fördel och kunde användas som ”wireframes” går i linje med Brown (2011) som menar att ”wireframes” underlättar utvecklingen av bland annat applikationer.

Då den redan utvecklade applikationen har används som ”wireframes” och på så sätt sparat vid överföringen, kan det anses relevant att titta på hur lång tid det har tagit att utveckla applikationen.

Det som är väsentligt i sammanhanget är att databasens utformning, applikationens affärslogik samt applikationens GUI har tagit cirka ett år att utveckla i samråd med de som använder applikationen. Att det har tagit cirka ett år att utveckla applikation har stor grund i anpassning gentemot användaren och utvecklingen av lämpligt användargränssnitt samt bakomliggande affärslogik.

Med bakgrund till detta är det inte rimligt att anta att utvecklingen av applikationen hade gått mycket fortare om det hade börjat utvecklats i ett ramverk (det ska dock understrykas att detta inte har studerats i denna uppsats och är endast en bedömning). Testningen och dialogen med användaren är av största vikt för att applikationen ska bli ändamålsenligt, vilket tar tid, detta menar även Guimaraes et al. (2004) som hävdar att det är viktigt med dialog mellan utvecklare och tilltänkta användare.

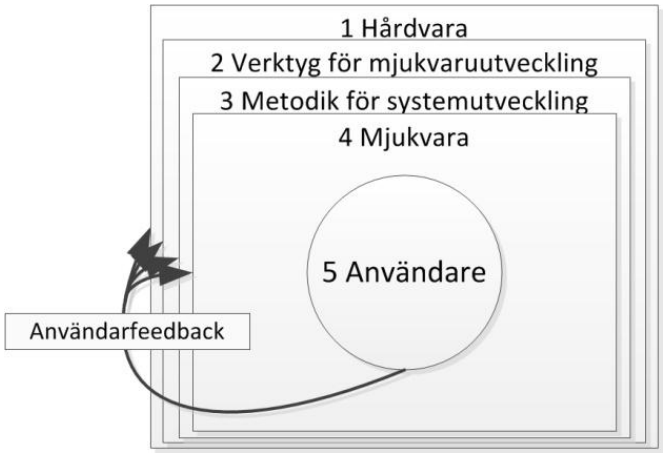
5.3 Mjukvaruutvecklingens förutsättningar

Ur ett vidare perspektiv syftar överföringen till att skapa förutsättningar för att kunna tillmötesgå användares krav och behov samt att skapa en bättre och konkurrenskraftigare produkt. Uppsatsen har visat att det är till fördel att använda sig av ramverk även om applikationen redan är utvecklad, behovet kan dock variera pågrund av vissa faktorer, vilka diskuterats ovan.

Men det måste finnas fler faktorer än att bara använda sig av ramverk för att lyckas skapa en konkurrenskraftig applikation. Ramverk som är en produkt av ”best practice” och erfarenheter måste utvecklats efter ett specifikt behov som har uppstått. Frågan är vilka behov och möjligheter som i slutändan skapar en användbar applikation, det är ju inte bara ett ramverk. Utifrån detta kan frågan ställas om vilka andra beståndsdelar som är nödvändiga vid mjukvaruutveckling för att bygga en ändamålsenlig applikation. I denna uppsats har enligt min mening tre beståndsdelar gällande mjukvaruutveckling berörts.

Den första beståndsdelen rör verktyg för mjukvaruutveckling vilket i denna uppsats har utgjorts av ramverket ”Yii”. Den andra beståndsdelen rör metodik för systemutveckling där ”toFramework refactoring” har används i denna uppsats. Den sista beståndsdelen som har berörts i denna uppsats är mjukvaran i sig det vill säga den överförda applikationen.

Ytterligare två beståndsdelar har verifierats: hårdvara och användare. Användare utifrån Guimaraes et al. (2004) som menar att användaren är viktig vid mjukvaruutveckling och hårdvara som får ses som en förutsättning för mjukvara. Dessa fem beståndsdelar illustreras i figur 10 nedan. Figuren illustrerar även hur dessa kan ses som sammanhängande, vilket kommer diskuteras vidare nedan.



Figur 10. Ramar för användaren

1. Hårdvara – Från hertz till gigahertz: Som redan fastslogs i början av 1960 talet när begreppet *”software crisis”*[16] myntades har utvecklingen gällande hårdvaran och dess prestanda möjliggjort att väldigt stora program kan skrivas. Kort sagt, desto kraftfullare hårdvara som finns att tillgå desto mer komplexa och krävande system kan byggas. Detta skapar förutsättningar åt verktyg för mjukvaruutveckling.

2. Verktyg för mjukvaruutveckling – Från DOS till ramverk: I ett försök att mästra de möjligheter som ges på grund av bättre hårdvara har bland annat ramverk, objektorienterad programmering och kodbibliotek utvecklats. Som antyds i rubriken ovan kan man enkelt uttryckt mena att tidig mjukvaruutveckling skedde i DOS-miljö med mycket enkla medel till att idag använda sig av mer avancerade verktyg såsom ramverk.

3. Metodik för systemutveckling – Från vattenfall till ”Scrum”: Samtidigt som nya verktyg för mjukvaruutveckling har utvecklats har även metodiken för att använda dessa utvecklats. Till exempelvis utvecklades den för tillfället populära metoden ”Scrum” inte förrän början av 1990 talet vilken kan ställas i förhållande till den enklare och äldre vattenfallsmodellen.

4. Mjukvara – från DOS till webb 2.0: Ett resultat av att hårdvara, verktyg och metodik har utvecklats är även att mjukvaran utvecklats. Från att användare tidigare arbetat i enkla lokala DOS-program blir det idag allt vanligare att använda sig av flexibla molntjänster, vilket har möjliggjorts av de faktorer som listats ovan.

5. Användaren – Brukaren av mjukvaran: Allt syftar till att konstruera en mjukvara som i förlängningen skapar förutsättningar för användaren. Utifrån krav och behov från användaren utvecklas hårdvara, verktyg, metodik och mjukvara. Detta i syfte att skapa mer dynamiska miljöer för användaren och större ramar för användaren att röra sig inom, ramar som gestaltas i figur 11.

[16] ”Software crisis”, har sin grund i att det är svårt att skriva korrekt, begriplig och kontrollerbar mjukvara. Den komplexitet ges möjlighet att öka i takt med att hårdvaran utvecklas.

5.3.1 Konsekvenser av större ramar

Utifrån användarens krav och behov kan det alltså anses rimligt att mer avancerad teknik och metodik fortsätter att utvecklas. Samtidigt blir de olika ramarna större och fler möjligheter ges att göra mjukvara som är mer ändamålsenlig och anpassad för användaren.

Med bakgrund till detta kan det vara intressant att ställa sig frågan vad kommande utveckling för webbapplikationer kommer ha att erbjuda. Något som är i ropet idag är webb 2.0 som introducerades av Tim O'Reilly år 2004 (O'Reilly, 2007). Tillsammans med molntjänster ställer webb 2.0 mycket högre krav på hårdvaran och mjukvaran för att generera användbarhet och dynamik för användaren. Tackvare utveckling inom samtliga ramar som diskuterades tidigare, kommer det antagligen skapas ytterligare möjligheter för användaren.

Utifrån webb 2.0 uppsving de senaste åren kan det anses osannolikt att nya applikationer utvecklas utan någon form av ramverk eller kodbibliotek. Att det skulle ske en utveckling där man går tillbaka till fristående kod anses alltså orimligt med tanke på de krav som ställs utifrån exempelvis webb 2.0 och molntjänster.

Ett annat mer förkommande fenomen är *”Content Management System” (CMS)*. Dagens CMS lösningar är i stort utformade så att organisationer ska kunna sköta sin egna webbplats med mycket enkla medel och programmeringskunskaper. Om man även tar detta ett steg längre blir det eventuellt möjligt att varje användare har ett gränssnitt som de kan redigera efter önskat behov. Genom ”drag and drop”[17] bygger användaren sin egna applikation eller användargränssnitt för respektive molntjänst. Alltså någon slags webb 3.0 där användaren bygger det dynamiska innehållet själv, då användaren inte längre nöjer sig med de fasta funktioner som finns inom applikationen.

[17] ”Drag and drop”, hantera virtuella objekt genom "greppa" dem och dra det till önskad plats inom aktuell applikation. ”Drag and drop” kan även användas för att påvisa relationer mellan olika objekt.

6 Slutsats

Den första frågan uppsatsen syftade till att besvara var: *Vilka komplikationer finns då en applikation baserad på fristående kod ska överföras till ett existerande ramverk?*

Gällande fråga ett så vittnar uppsatsen om att de komplikationer som fanns vid överföringen inte var av nämndvärd karaktär. Slutsatsen utifrån detta blir alltså att det inte finns några direkta hinder för att genomföra en överföring. Sammanfattningsvis är svaret, inga komplikationer, på fråga nummer ett.

Fråga nummer två var följande: *Är det förmånligt ur tidsaspekt och underhållsaspekt, att överge en redan fungerande applikation skriven i fristående kod för att istället skapa samma funktionalitet i ett ramverk?*

Det kan anses som en fördel att överföra en redan utvecklad applikation om det finns ett betydande behov av vidareutveckling och underhåll. Förutom tidsbesparningar vid vidareutveckling skapar även ramverket en viss struktur vilket programmeraren får anpassa sig till för att dra störst nytta av ramverkets funktionalitet. Ett resultat av detta blir att koden blir likformig och strukturerad vilket kan anses tidssparande isig då koden blir mer greppbar. Sammanfattningsvis är svaret ja, på fråga två.

Vidare slutsatser

Utifrån den tidigare diskussionen om webbapplikationers utveckling kan det anses rimligt att dessa kommer bli mer dynamiskt utformade i framtiden. I ljuset av detta är det troligt att det kommer bli allt viktigare med färdiga kodbibliotek och ramverk som innehåller grundläggande funktionalitet. Vad som innefattas av grundläggande funktionalitet kommer sannolikt utökas med tiden. Samtidigt är det troligt att utveckling av webbapplikationer där bara fristående kod används kommer bli för resurskrävande och således inte konkurrenskraftiga.

I uppsatsen användes metoden ”toFramework refactoring”. Metod har under studien upplevts ändamålsenlig och fyllt en funktion. En slutsats som kan dras är att studien inte hade genomförts lika metodiskt utan denna metod. Däremot hade det varit intressant att testa denna metod ytterligare, vilket även är ett förslag till vidare forskning. Gällande vidare forskning kan det även anses intressant utifrån studiens resultat att undersöka kostnadsestimeringsmodeller vid användning av ramverk avsedda för webbapplikationer.

7 Källor

Araújo , R. Adriano, S. Oliveira, L.I. (2011) *Hybrid morphological methodology for software development cost estimation,* Informatics Center, Federal University of Pernambuco, Recife, PE, Brazil

Beedle, M. Scwaber, K. (2002) *Agile Software Development with "SCRUM"*, Prentice Hall

Brown, D. (2011) *Communicating Design: Developing Web Site Documentation for Design and Planning, Second Edition*, New Riders

D'Hondt, M. Fernández-Ramil, J. Guehéneuc, Y. Mens, T (2010) *Guest Editors' Introduction: Software Evolution,* IEEE Software, Vol. 27, No. 4. (July 2010), pp. 22-25

Ferraiolo, D. Kuhn, R. (2007) *Role-based access control (2nd Ed.)*, Antech House

Forsberg, C. Wengström, Y. (2008) *Att göra systematiska litteraturstudier*, Författarna och Bokförlaget Natur och Kultur, Stockholm

Fowler, M. (1999) *Refactoring. Improving the Design of Existing Code*, Addison-Wesley

Guimaraes, T. Staples, S. McKeen, J. (2004) *Important Human Factors for Systems Development Success: A User Focus*, in "Strategies for Managing IS/IT Personnel" edit by Magid Igbaria and

Hearn, J. Fuer, D. Higginson, IJ. Sheldon, T. (1999) *Systematic reviews*, Palliative medicine 1999;13:75-80

Mens, T. Tourwé, T. (2004) *A Survey of Software Refactoring*, Universiteit of Brussel

O'Reilly, T. (2007) *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software,* O'Reilly Media, Sebastopol (CA) USA

Reenskaug, T. (2003) *The Model-view-controller (MVC) Its Past and Presents*, University of Oslo

Riehle, D. (2000) *Framework Design: A Role Modeling Approach*, Ph.D. Thesis, No. 13509. Switzerland, ETH Zürich

Riehle, D. Gross, T. (1998) *Role Model Based Framework Design and Integration*, (OOPSLA '98). ACM Press,. Page 117-133

Schwaber, K. (1999) *"SCRUM" Development Process, Advanced Development Methods*, Burlington

Schwaber, K. Sutherland, J. (2010) *"SCRUM"*

Seacord, R. Plakosh, D. Lewis, G. (2003) *Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices*, (SEI Series in Software Engineering). Addison-Wesley.

Veerraju, R.P.S.P. Srinivasa, R. Murali, G. (2010) *Refactoring and Its Benefits*, International Conference on Modeling, Optimization and Computing (ICMOC 2010)

Winsett, J. (2010) *Agile Web Application Development with Yii1.1 and PHP5*, Birmingham – Mumbai