UPTEC STS 06 020
Examensarbete 20 p
Januari 2007

Java, ramverk och komplex webutveckling

En helhetsstudie av utvecklingsprocessen bakom webplatsen Oddsnews.com

Jacob Orrje
Dennis Persson

UPPSALA
UNIVERSITET

Teknisk- naturvetenskaplig fakultet
UTH-enheten

Besöksadress:
Ångströmlaboratoriet
Lägerhyddsvägen 1
Hus 4, Plan 0

Postadress:
Box 536
751 21 Uppsala

Telefon:
018 – 471 30 03

Telefax:
018 – 471 30 00

Hemsida:
http://www.teknat.uu.se/student

Abstract

Java, ramverk och komplex webbutveckling

Java, frameworks and complex web development

Jacob Orrje, Dennis Persson

This thesis explores the development process of a highly complex, dynamic website, www.oddsnews.com. We first evaluate different methodologies for software development, and find that an Agile approach is suitable for this particular project. The web application is developed in Java using, among others, the Spring framework and the Hibernate Object-to-relational mapping tool. Much effort is put into writing the application as flexible and extensible as possible, for example by using Springs ”Dependency Injection” capabilities. We discuss how the application automatically can gather structured as well as unstructured data from a lot of different sources and present this to the user in an easily understandable way.
We then evaluate the project so far and find that the choice of technology has helped Oddsnews to work according to the Agile principles. We can also see that our broad STS competence has helped the project by allowing us to focus on other aspects as well as the purely technical.

Handledare: Fredrik Lydén
Ämnesgranskare: Olle Eriksson
Examinator: Elisabet Andrésdóttir
ISSN: 1650-8319, UPTEC STS 06 020

Populärvetenskaplig sammanfattning

Hur går det egentligen till att utveckla en stor avancerad webbsida med mycket information? Internet är ett tekniskt område som förändras i snabb takt. Nya tillämpningar av tekniken tillkommer och företag kommer på nya idéer på hur man kan använda sig av Internet för att öka sin lönsamhet eller etablera en helt ny verksamhet. Trots teknikens otaliga användningsområden är ändå information i någon form det centrala för alla Internetföretag. För företag verksamma på Internet är det därför viktigt att utveckla applikationer för insamlande, bearbetning och presentation av information.

Idag finns det många teknologier för att utveckla komplexa applikationer som bearbetar data på ett högt automatiserat sätt – Suns Java och Microsofts .NET är kanske de vanligaste. Men hjälp av dessa kan man skriva en applikation som själv samlar in data från andra delar av Internet, lagrar informationen, utvinner ny information ur informationen med hjälp av statistiska metoder samt presenterar den nya informationen på för användarna bekvämt sätt. För ett företag verksamt inom Internet är sådana metoder mycket användbara, eftersom man inte behöver ha avlönad personal för att göra samma uppgifter. Inom ramen för vårt examensarbete har vi deltagit i utvecklingen av en sådan applikation, skriven i Java, åt företaget Oddsnews Ltd. Applikationen skapar data till en webbsida som jämför odds mellan spelbolag och presenterar relevant statistik och relevanta nyheter till sina besökare. Resultatet av utvecklingen kan ses på webbsidan www.oddsnews.com. Det som har intresserat oss är hur de metoder man använt under utvecklandet av applikationen har påverkat hur väl applikationen fungerar när den är färdig. Vi har tagit ett helhetsperspektiv och försökt relatera så skilda delar som valet av tekniska lösningar och designbeslut, organisationen av utvecklingslaget samt affärsmässigheten till varandra.

Vad vi upptäckte genom vårt arbete var att valet av tekniska lösningar ofta påverkar utvecklingens organisatoriska förutsättningar. Traditionellt har man snarare pekat på behovet av att ha ett välorganiserat utvecklingslag för att få bra kod, varför det var intressant att se hur välorganiserad kod även kan kompensera för mer löst organiserad utvecklingsprocess. Exempelvis gjorde det att olika delar av koden var relativt oberoende från varandra att de blev enklare att ha utvecklare boende på olika orter verksamma i projektet.

Vårt arbete har även bestått i att utvärdera de lösningar vi valt. Vi diskuterar de tekniska lösningarna, användarvänligheten av materialet som presenteras och möjligheten för företaget att tjäna pengar på applikationen. Kortfattat kan man se att en automatiserad applikation för informationsbearbetning har mycket låga driftskostnader, men att det behövs stora resurser för att marknadsföra produkten. Vi kunde också se att det som Internetföretag ofta är svårt att knyta an till sina kunder, eftersom man inte har någon personlig kontakt med dem. Alternativet till detta är att utvärdera användarvänligheten i applikationen genom statistiska data över besökarna.

Att arbeta med avancerad webbutveckling har varit en lärorik verksamhet och man blir ofta överraskad av komplexiteten bakom de enkla webbsidor man som användare dagligen använder. Att nå lönsamhet som Internetföretag är inte något gjort i en handvändning, men förhoppningsvis kommer vår analys att bli en bra grund för företaget Oddsnews Ltd för att nå detta mål.

Förord

Detta examensarbete är resultatet av det utvecklingsarbete som bedrivits åt företaget Oddsnews Ltd mellan maj-oktober 2006. Oddsnews Ltd är ett nybildat företag som arbetar med webbaserade metoder för att jämföra odds mellan olika spelbolag samt att förmedla sportrelaterad information och statistik. I och med detta examensarbete har även vi rapportförfattare tagits in som minoritetsägare i företaget. Vårt arbete har omfattat både praktisk webbutveckling i java och studier av designmönster och utvecklingsprocesser. Följande rapport syftar till att knyta samman dessa praktiska och teorietiska erfarenheter till en helhet – genom att anlägga ett brett systemperspektiv på en mycket konkret utvecklingsprocess.

Tack

Vi vill tacka alla de som gjort detta examensarbete möjligt. Främst Fredrik Lydén hos Oddsnews Ltd för en ovärderlig hjälp med det konkreta utvecklingsarbetet samt handledning med rapporten. Dessutom vill vi även tacka de andra på företaget som kommit med intressanta tips och idéer: Martin Ekström och Robert Högström. Tack även Olle Eriksson vid Institutionen för Informationsteknologi som tog på sig rollen som ämnesgranskare av examensarbetet. Slutligen: tack till våra vänner, bekanta och familjer och alla andra som har stöttat oss under vårt arbete.

Innehållsförteckning

1. Inledning

I takt med att Internet har vuxit har en hel del nya Internettjänster tillkommit. Tjänsternas komplexitet har med tiden ökat och har gått från att ha varit statiska sidor till att allt mer utgöras av komplexa dynamiska applikationer. När man idag utvecklar en komplex webbapplikation måste man ta hänsyn till en rad olika parametrar. Hur är applikationens prestanda? Hur utbyggbar och lättunderhållen är kodbasen på vilken applikationen vilar? Vilken är målgruppen för webbapplikationen och är informationen tillräckligt väl presenterad för att locka besökarna att komma tillbaka efter sitt första besök? Dessa är bara några av de frågor man måste ta ställning till under utvecklingen av en modern webbapplikation. Många av frågorna hänger ihop med varandra och för att kunna besvara dem behövs ett helhetsperspektiv på hela utvecklingsprocessen. Vi måste fråga oss vilken information det är vi vill förmedla via webbapplikationen, hur vi ska få tag på denna information, hur vi ska lagra den på ett effektivt sätt, hur vi ska presentera den för besökarna samt inte minst: hur applikationen ska designas och skrivas som ska hantera och knyta ihop alla dessa olika aspekter av webbapplikationen.

Idag finns det många färdiga lösningar med öppen källkod som man kan använda vid utvecklingen av en webbapplikation. Dessa lösningar kan avhjälpa arbetsbördan i alla delar av en utvecklingsprocess, men deras viktigaste funktion är kanske att utgöra en stabil standardiserad grund varpå applikationen vilar. Dessa lösningar avhjälper mycket onödigt arbete och kan snabba på utvecklingen avsevärt men de ställer dock även större krav på kompetens i inledningen av utvecklingsarbetet – främst då kring vilka av dessa lösningar som är bättre än andra, vilka som fortfarande utvecklas och vilka som hjälper och inte stjälper webbapplikationens utveckling. I och med att vi under vårt arbete i hög utsträckning har använt oss av lösningar med öppen källkod blir en viktig del av denna rapport en presentation av vilka lösningar som vi använder samt varför vi valt dem framför andra.

Denna rapport är resultatet av utvecklingsarbetet med applikationen bakom webbtjänsten Oddsnews.com. Applikationen har skrivits med Java med Spring som ramverk. Detta arbete har väckt många tankar kring den process som en utveckling av en komplex webbapplikation utgör. Eftersom vi rapportförfattare har varit delaktiga i stort sett alla delar av utvecklingsarbetet har vi sett vilket behov det finns av samverkan mellan olika delar av utvecklingen – exempelvis att det finns ett behov av att se till samma övergripande mål vid valet av programvara, designmodell och webblayout. Behoven hos vår uppdragsgivare – Oddsnews Ltd har varit att snabbt få ut en produkt på marknaden som attraherar en bredare sportintresserad allmänhet. Eftersom tjänsten är ny har man prioriterat en snabb utvecklingstakt och en lättutbyggd kod framför en optimerad applikation. Vad man vill ha är en applikation som kan utvecklas i takt med att besökarantalet växer och deras krav blir tydligare. Eftersom Oddsnews Ltd är ett nystartat företag med relativt små resurser har det även varit ett krav att lösningen ska vara mycket kostnadseffektiv. Företagets storlek har bidragit till att det har varit enklare att anta ett helhetsperspektiv på utvecklingsarbetet, något som även har påverkat valet av frågeställningar i denna rapport. Under vårt arbete har vi kommit i kontakt med i stort sätt alla företagets verksamhetsområden och just därför har det varit lätt att dra paralleller mellan exempelvis företagsekonomiska behov och de krav som ställs på utvecklingsprocessen och detta påverkar även såklart det perspektiv vi anlägger i studien. Under arbetets gång har vi också på ett ytterst konkret sätt fått en insikt i

realiteten för små svenska entreprenörer inom IT-branschen och de krav som denna realitet ställer på alla verksamhetsområden.

Det är vår erfarenhet att man i studier av teknikens förhållande till enskilda människor eller till samhället tenderar att antingen anlägga ett smalt tekniskt fokus där man försöker se människor som delar i tekniska system (ett tydligt exempel på detta är användargränssnittsstudier) eller att anlägga ett brett perspektiv där tekniken behandlas teoretiskt, exempelvis som en produkt på en marknad. Vår personliga övertygelse är att det är gynnsamt för vår studie att få en koppling mellan dessa två synsätt – att kombinera en djup teknisk förståelse med ett helhetsperspektiv som tar in mer än tekniska faktorer. Genom att göra det kan vi på ett mer konkret sätt exempelvis se hur valet av tekniska lösningar ger följdverkningar inom andra delar av utvecklingen än de rent tekniska.

Rapporten kan ses som en fallstudie på en väl avgränsad utvecklingsprocess hos ett litet kunskapsintensivt företag. Genom denna konkreta avgränsning kan vi studera utvecklingen både på bredden och på djupet på det sätt som vi redogör för ovan. Arbetet har dock även styrts av de förutsättningar som våra uppdragsgivare har satt upp. Utgångspunkterna för detta har varit en uppsättning egenskaper vilka varit: utbyggbarhet framför prestanda, användarnas reella behov framför optimering samt målgruppsfokus redan på planeringsstadiet och utvecklingsstadiet.

1.1 Syfte

Syftet med detta examensarbete är att ta ett helhetsgrepp på en utvecklingsprocess för att kunna se kopplingarna mellan dess olika delar samt hur dessa var och en för sig och tillsammans påverkar den slutgiltiga produkten. Genom att delta i utvecklingen av webbtjänsten Oddsnews.com åt företaget Oddsnews Ltd och samtidigt studera detsamma som en fallstudie av ett innovationssystem vill vi besvara frågeställningen: ”Hur påverkar de olika delarna av utvecklingsprocessen av en komplex webbapplikation denna applikation som en färdig produkt”.

2. Företaget Oddsnews Ltd[1]

I följande kapitel ger vi en översiktlig bild av det företag examensarbetet skrivits åt, Oddsnews Ltd. Vi beskriver företagets affärsidé, resurser och strategier – vilket har påverkat inriktningen på vårt arbete.

Många av de riktlinjer som funnits för examensarbetet kan härledas till de speciella förhållanden som funnits hos Oddsnews Ltd åt vilket examensarbetet har skrivits. Företaget Oddsnews Ltd startades under våren 2006 av tre svenska entreprenörer varav två har en bakgrund som konsulter inom IT-branschen och den tredje arbetar inom fastighetsbranschen. De två IT-konsulterna har ansvar för utvecklingen – vilket har varit det som krävt mest resurser under den tid detta exjobb har vuxit fram (maj-okt 2006). Den tredje entreprenören har fungerat som ekonomiskt ansvarig och har hanterat kontakter med exempelvis banker. Företaget är registrerat i England trots att grundarna är verksamma i Sverige, detta eftersom grundarna anser att den svenska spelmarknaden är relativ osäker och att de regler som gäller för företag som själva inte är spelbolag men har en koppling till branschen är otydliga. Dock har företaget kontor i Farsta i Stockholm och de riktar sig till en svensk marknad.

2.1 Affärsidén

Företagets affärsidé består i att fungera som en prisjämförelsetjänst – men som jämför spelbolagens odds istället för exempelvis konsumentpriser. Spelmarknaden på Internet består idag av ett stort antal aktörer och för konsumenten kan det vara mycket svårt att skilja dem åt – därför ansåg man att det fanns ett behov av en webbtjänst som kunde tydliggöra skillnaderna mellan spelbolagen och som förenklade en jämförelse av vilka odds de gav spelaren. Redan finns dock en uppsjö liknande tjänster som just jämför spelbolagens odds[2]. Dock är ingen av dessa tjänster på svenska – vilket grundarna av Oddsnews Ltd ansåg vara en marknadsnisch de kunde fylla. I huvudsak har man planerat att få inkomster genom att låta spelbolagen annonsera på webbsidan både genom grafiska banners och genom integrerade länkar från själva oddstipsen. I nuläget har flertalet av de i Sverige verkande spelbolagen så kallade Affiliate-program vilket i ett inledningsskede ses som det bästa sättet att få intäkter genom webbtjänsten.

2.2 Resurser

Oddsnews Ltd får i många avseenden anses vara ett projekt som grundarna driver vid sidan om sina vanliga karriärer. Man har inte någon extern finansiering – vilket innebär att den huvudsakliga resursen företaget har är de inblandades erfarenhet av liknande IT-projekt. Företagets knappa ekonomiska resurser ställer extra höga krav på utvecklingsprojektet – där behovet av att snabbt komma från en konceptuell idé till en färdig produkt blir viktigt. Detta ställer höga krav på en välorganiserad och snabbrörlig utvecklingsprocess – där en utbyggbar design och snabb utveckling är viktigare än en optimerad kod. Företagets ekonomiska verklighet ställer alltså upp ramar för den rent tekniska utvecklingsprocessen – något man varit tvungen att ta hänsyn till från första stund. Något som ytterligare har bidragit till behovet av att snabbt komma ut med en

[1] Informationen i följande stycke kommer om inget annat anges från samtal med grundarna av Oddsnews Ltd.

[2] För en sammanställning av liknande webbtjänster, se http://www.braodds.com/ (2006-09-29)

produkt är att den svenska spelmarknaden är mycket dynamisk, där ständigt nya aktörer tillkommer och andra försvinner. Man har insett att det inte är särskilt otroligt att något annat företag snart kommer med en liknande idé – vilket på grund av Oddsnews Ltd:s begränsade resurser skulle innebära ett stort konkurrensproblem. Vad man vill göra är alltså att få ut en produkt för att etablera sig på marknaden och därigenom förekomma potentiella konkurrenter. Genom att tidigt komma ut med en prototyp har man också hoppats på att kunna få finansiering från ekonomiskt starkare aktörer.

2.3 Strategi

Det centrala för webbtjänster i allmänhet och Oddsnews i synnerhet är information i någon form. Ett nödvändigt men inte tillräckligt villkor för en webbsidas framgång är att den har någonting att förmedla till sina besökare som dessa är intresserade av. Detta skulle kunna vara informationen om sortimentet i en webbutik eller den information som förmedlas mellan deltagarna i ett Community.

För ett företag som skapar en webbtjänst blir alltså en mycket central del i dess affärsstrategi hur den ska få tillgång till den information som den ska förmedla. Här kan man ha flera olika strategier. Man kan välja att skapa informationen själv – vilket har fördelen att webbtjänsten har större chans att innehålla unik information och att därför bli intressantare för besökarna. Som företag kan man dock vara tvungen att avlöna skribenter som skapar informationen – vilket kan bli dyrt för ett litet nyuppstartat företag. En alternativ strategi som blivit populär på senare tid är att knyta en så kallad Community-känsla till sin webbtjänst. Det handlar om att få själva användarna att känna en tillhörighet till webbtjänsten och de andra användarna och att användarna därigenom själva skapar webbsidans information. Det finns många exempel på företag som lyckats med denna strategi och som lyckats växa sig stora med hjälp av den. För en nystartad webbtjänst kan dock denna strategi innebära något av ett moment 22 – om tjänsten inte har några användare finns det ingen intressant information för användarna och därmed får man inte heller några nya användare. För att lyckas med denna strategi kan det alltså i ett inledningsskede vara intressant att kombinera den med någon annan.

Frågan är då vilken strategi som är fruktbar för ett litet företag som Oddsnews Ltd. Har man varken resurser att skapa tillräckligt mycket unik information eller tillräckligt med användare för att låta dem skapa informationen på sidan själva kan en alternativ strategi vara att använda sig av information som redan finns på Internet. Detta blir oftast betydligt billigare än att skapa informationen själv – även om denna strategi också medför vissa problem. Det tydligaste problemet är att denna information inte blir unik – eftersom samma information redan finns på andra, kanske redan mer etablerade, delar av Internet. Detta kan lösas genom att presentera och relatera informationen man hämtat på nya sätt – och på så sätt placera informationen i ett sammanhang som gör att informationen på webbtjänsten ändå blir de facto unik. Detta kan göras genom att relatera information från flera skilda delar av Internet på samma sida – och på så sätt koppla ihop information på ett sätt som skulle vara mycket svårt för användaren själv att göra. Ett annat problem med att återskapa information kan vara de juridiska frågor det väcker. Som företag är det därför ytterst viktigt att hämta information från sådana källor man får återskapa. Detta blir i många fall en avvägningsfråga – eftersom det inte alltså är helt tydligt hur man ska tolka sådant som upphovsrätt och citaträtt på Internet.

3. Metod för utvecklingsprocessen

Eftersom de förhållanden som rådde på Oddsnews Ltd är de typiska för ett litet företag med små resurser blir kraven på dess utvecklingsprocesser något annorlunda än hos större företag. Utvecklingsprocessen hos Oddsnews Ltd påverkas av att man inte har något funktionellt kontor eller formella hierarkier och att man varken har externa kunder eller hårda deadlines. Det har därför varit viktigt för oss och Oddsnews Ltd att hitta arbetsprocesser som är anpassade till och som förenklar denna situation – både organisatoriskt och tekniskt. I denna del redogör vi för olika modeller av utvecklingsprocesser och analyserar avslutningsvis hur den konkreta utvecklingen hos Oddsnews Ltd kan relateras till dessa modeller. Många av de modeller för mjukvaruutveckling vi redogör för nedan har ofta utvecklats med större etablerade företag i åtanke. Därför är det för oss intressant att utvärdera dessa i relation till Oddsnews situation. Vår hypotes är att Agila metoder passar bättre i ett mindre företag som Oddsnews men att även denna modell måste anpassas till företagets konkreta situation.

3.1 Modeller för mjukvaruutveckling

Programmering och systemutveckling har länge setts som något av en svart konst, där det inte finns några pålitliga metoder för hur man uppnår framgång i ett projekt. Ett projekts utfall berodde istället på de enskilda programmerarnas genialitet och förmåga att tänka i nya och oväntade banor. Detta är en del av den kultur som programmering har omgetts av sedan de första så kallade ”hackarna” verkade på MIT i USA på 1960-talet[3].

Men på samma sätt som den tidiga ingenjörskonsten gick från ett oorganiserat trial-and-error”-tänkande till en systematisk vetenskap[4] så blev även trycket på datavetenskapen större och större att utveckla en metodologi som kunde användas om och om igen vid utveckling. Mcgee beskriver hur skeppsbyggarkonsten under 1800-talet utvecklas från ett hantverk (Craftmanship) till att handla om något som snarast skulle kunna översättas till ingenjörskonst (Draftmanship). Allt eftersom systemen växte, både i antal och till omfång, blev det helt enkelt omöjligt att utveckla system utan att ha en grundläggande plan i bakgrunden. Programmeraryrket har alltså sedan sin tillkomst genomgått en process som i mycket liknar den som Mcgee beskriver: man har genomgått en professionaliseringsprocess där informella koder till viss del ersatts med systematiska, klart definierade metoder.

I detta arbete har vi valt att fokusera på ett fåtal av dessa olika formella metoder. Detta för att visa hur den metod för utveckling som vi slutligen valde att arbeta efter har influerats av tidigare metoder. Metoderna kommer att tas upp i kronologisk ordning och därför inleds avsnittet med den så kallade ”Vattenfallsmodellen” från 1970-talet för att sedan gå vidare till den så kallade RUP-modellen (1990-tal) fram till det som är på modet hos många utvecklare idag: så kallade Agila utvecklingsmetoder.

---

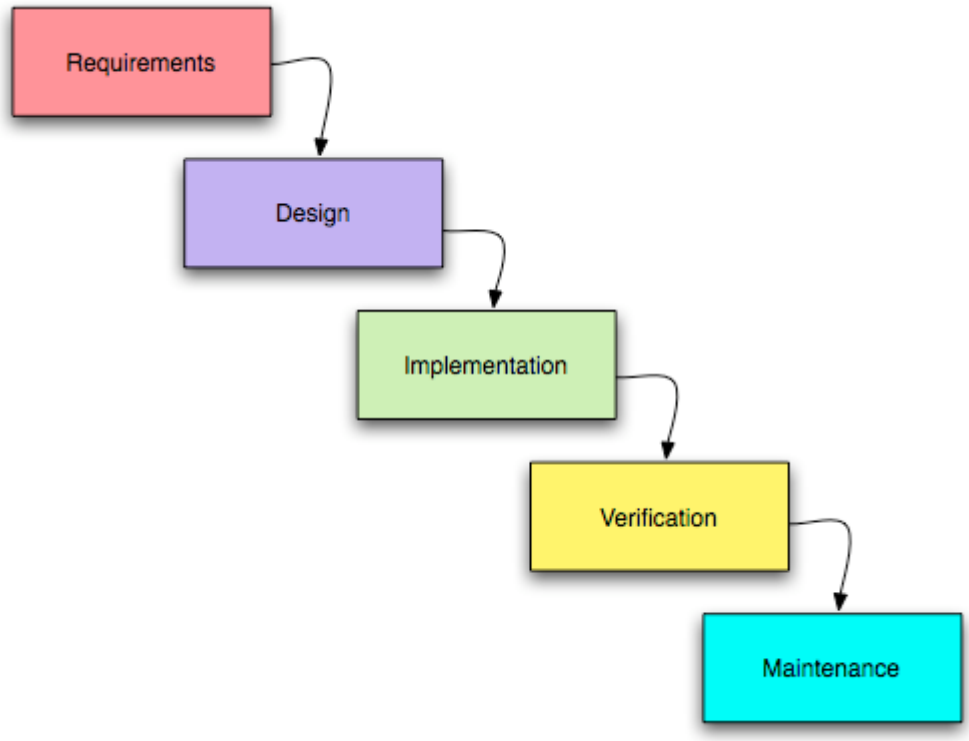
[3] Raymond, Eric S., *”A brief history of Hackerdom”,* http://library.n0i.net/advocacy/hacker-history/, 2006-10-13

[4] Mcgee, David, “From Craftsmanship to Draftsmanship: Naval Architecture and the Three Traditions of Early Modern Design”, *Technology and Culture* 40.2 (1999) 209-236

3.1.1 Vattenfallsmodellen

Den så kallade Vattenfallsmodellen för programvaruutveckling tillskrivs ofta den amerikanska forskaren W.W. Royces och mer specifikt en artikel av honom från 1970[5]. Det något ironiska i sammanhanget är att Royce i artikeln använde modellen som ett varnande exempel för hur utvecklingen *inte* ska gå till och han använde heller aldrig termen ”waterfall” i artikeln. Däremot har vattenfallsmodellen haft, och har fortfarande, ett stort inflytande på systemutveckling, kanske främst för att den i vart fall på ytan väldigt tydligt och på ett enkelt sätt definierar de steg som ska följas (se figur 1). Den beskriver en utvecklingsprocess som på papperet är väldigt välplanerad och organiserad, vilket har uppskattats mycket, inte minst på ledningsnivå inom organisationer som arbetar med mjukvaruutveckling.

Vattenfallsmodellen delar upp utvecklingsprocessen i ett i förväg bestämt antal delar, där var och en av dessa delar följer linjärt på den föregående. Arbetet på en senare del i utvecklingen ska inte påbörjas förrän föregående steg har slutförts. När arbetet på en av dessa delar slutförts ses utvecklingsfasen som färdig och dess resultat ändras inte senare i utvecklingsprocessen. Själva kodskrivandet kan inte heller det börja förrän alla kravspecifikationer har samlats in och behandlats noggrant, detta för att undvika att kraven senare visar sig vara felaktiga, eftersom detta skulle leda till dubbeljobb.



Figur 1 - Schematisk modell över vattenfallsmodellen[6]

I början av ett projekt som arbetar enligt vattenfallsmodellen läggs stor energi ner på att ta fram de krav som systemet och applikationen kräver. Ofta skapas stora mängder

[5] *Royce, Winston,* "Managing the Development of Large Software Systems", *Proceedings of IEEE WESCON,* vol. 26 (1970), *1-9*

[6] http://en.wikipedia.org/wiki/Image:Waterfall_model.png, 2006-09-08

dokumentation över kravspecifikationer och liknande som sedan följs under resten av utvecklingsprocessen. Tidsplaner och arbetsfördelning slås även det fast i början av projektet med utgångspunkt i specifikationen. Framgången för projektet utvärderas sedan i senare steg efter hur väl processen följer den ursprungliga planen.[7]

Efter att kraven slagits fast går man vidare till designfasen. Här utarbetas modeller för hur designen ska göras, utifrån de krav som slogs fast i steg 1. Först när detta steg är klart börjar den faktiska koden skrivas av programmerarna. I slutet av implementationssteget integreras de olika delarna som eventuellt finns i systemet. Dessa olika delar har tidigare utvecklats helt oberoende av varandra, men eftersom de ska följa de designregler som slogs fast i steg två är tanken att de utan problem ska kunna integreras med varandra. Efter att implementering och integrering slutförts går man vidare till att testa systemet, för att vara säker på att det uppfyller specifikationerna. När alla buggar har rättats till är systemet klart för leverans till kund. Som ett sista steg följer sedan underhåll och eventuell vidareutveckling.[8]

3.1.2 Argument för och emot vattenfallsmodellen

Som antyddes i början av avsnittet har vattenfallsmodellen under sina år dragit på sig en hel del kritik, både från forskare och från näringsliv. Trots detta är den paradoxalt nog fortfarande ofta använd runt om i världen, varför är det så? Efter att ha ställt upp några ofta förkommande argument för och emot vattenfallsmodellen försöker vi ställa upp ett svar på detta.

Ett argument för vattenfallsmodellen är att den helt och hållet är linjär och har ett flertal klart definierade steg, det är en enkel process att lära sig. Det är också enkelt att tilldela personal och att planera, eftersom olika kompetenser behövs i olika steg. Mjukvaruarkitekter behöver kanske till exempel bara vara med i de två första stegen av utvecklingen. Dessutom gör modellen det lätt att ställa upp mått för hur projektet fortlöper. Genom att man har ställt upp en detaljerad plan i början kan man hela tiden se hur verkligheten förhåller sig till denna.

De argument som finns mot vattenfallsmodellen utgår inte främst från modellen i sig, utan hur verkligheten ser ut och att modellen appliceras på fel typ av projekt. För projekt där alla krav är fastställda i början av projektet, och sedan inte ändras under projektets gång kan vattenfallsmodellen fungera väl. Problemet är, argumenterar motståndarna, att verkligheten idag för systemutvecklare är att krav och förutsättningarna förändras hela tiden, och att det är omöjligt att känna till alla dessa i förhand. En av de mest inflytelserika artiklarna som argumenterar på detta sätt är Parnas och Clements ”A Rational Design Process: How and Why to fake it”[9].

> This paper brings a message with both bad news and good news. The bad news is that, in our opinion, we will never find the philosopher's stone. We will never find a process that allows us to design software in a perfectly rational way. The good news is that we

[7] Gibbs, R. Dennis, Project Management with the IBM® Rational Unified Process®: Lessons from the Trenches, IBM Press, 2006
[8] Gibbs, R. Dennis
[9] Parnas & Clements, ”A rational design process: How and why to fake it” http://www.ece.utexas.edu/~perry/education/360F/fakeit.pdf, 2006-10-20

> can fake it. We can present our system to others as if we had been rational designers and it pays to pretend to do so during development and maintenance.[10]

Parnas och Clements menar alltså att alla processer med nödvändighet blir en idealisering av verkligheten, men att de trots allt fyller ett syfte och därmed har ett existensberättigande. Just vattenfallsmodellen ses som den värsta av dessa idealtyper.

3.1.3 Utvärdering av vattenfallsmodellen

Vattenfallsprocessen verkar alltså lämpa sig främst till projekt som har specifikationer och krav väldigt tydligt definierade från början. Den lämpar sig också mer för större projekt med många utvecklare, där det krävs en stark disciplin och ledning för att få alla att arbeta mot samma mål. Vattenfallsmodellen är även användbar för utvecklingen av missionskritiska system, där utförlig dokumentation och specifikationer är helt nödvändiga för att t.ex. få berörda myndigheters godkännande.

Det står därmed tydligt klart att Oddsnews är ett projekt som inte lämpar sig för en process präglad av vattenfallsmodellen. Det som krävs i utvecklingen av Oddsnews är snabbhet och flexibilitet, att snabbt kunna ändra och bygga om funktioner utifrån användarnas krav. Utvecklarna av Oddsnews har inte heller tillgång till några konkreta användarkrav, utan dessa bedöms indirekt efter andra kriterier (en problematik vi utvecklar i avsnitt 6.2). Oddsnews består av ett litet team utan stora resurser, vilket även det minskar fördelarna av att arbeta enligt vattenfallsmodellen.

3.1.4 Rational Unified Process (RUP)

En annan viktig modell för utvecklingsprocesser är Rational Unified Process (RUP). RUP kan ses som en reaktion på vattenfallsmodellen och blev snabbt en av de absolut mest inflytelserika (om inte *den* mest inflytelserika) av de olika modellerna för utvecklingsprocesser.

RUP utvecklades av företaget Rational Software under det tidiga 1980-talet, men det fick inte sin slutliga benämning förrän 1995. Då köpte Rational upp det svenska företaget Objectory AB, vilket gjorde att de som senare skulle kallas ”the three amigos”, arbetade inom samma företag: James Rumbaugh, Grady Booch och Ivar Jacobsson. Resultatet av arbetet blev dels det som idag kallas UML (Unified modelling language), en standardiserad metod för att grafiskt beskriva ett programvarusystem, samt RUP.[11]

RUP kan sägas vara en induktiv metod i det att den utgick från de erfarenheter som Rational Software fick från sina många uppdrag, bland annat för den amerikanska militären. Genom att studera misslyckade projekt kunde man ställa upp ett antal ”best practices” för utveckling som sedan legat till grund för många utvecklingsprojekt.

1. Develop iteratively.
2. Manage requirements.
3. Use component architectures.
4. Model visually.
5. Continually verify quality.

---

[10] Ibid, s.1
[11] http://en.wikipedia.org/wiki/Rational_Software, 2006-11-05

6. Manage changes.[12]

Dessa sex ledord lade grunden för RUP, fram tills 2005 då en ny version av dessa lades fram av Rational, som nu ägs av IBM:

1. Adapt the process.
2. Balance competing stakeholder priorities.
3. Collaborate across teams.
4. Demonstrate value iteratively.
5. Elevate the level of abstraction.
6. Focus continually on quality.[13]

Förutom dessa ledord består RUP av flera andra delar som tillsammans bildar ett komplett ramverk för mjukvaruutveckling. RUP är inte tänkt som en färdig process på samma sätt som vattenfallsmodellen, utan varje projekt förväntas att välja ut de delar ur RUP-ramverket som ses som en tillgång för just det projektet. RUP fokuserar därmed mer på anpassningsbarhet till det aktuella projektet.

I RUP-modellen delas ett projekt in i fyra faser: Inception, Elaboration, Construction samt Transitions-fasen. Den stora skillnaden jämfört med vattenfallsmodellen är att dessa faser under RUP inte ses som linjärt följande på varandra, utan den iterativa arbetsmetoden gör att man ibland återvänder till ett tidigare steg när man funnit ett bättre sätt att lösa problemen under dessa.

Inception-fasen fokuserar på att slå fast ett projekts uppdrag och avgränsningar och har som slutligt mål att slå fast affärsmässigheten i det föreslagna projektet. Detta görs genom ett nära samarbete med uppdragsgivaren och leder till att ett antal ”use-cases” ställs upp. Ett use-case är ett funktionellt krav, som till exempel kan vara att ”olika användare ska ha olika rättigheter i systemet”. Utifrån dessa planeras sedan de kommande iterationerna och alla användarkrav betas av inkrementellt.[14] Skulle det visa sig att projektet inte har möjlighet att bli lönsamt eller inte är genomförbart på grund av någon annan anledning (inte rätt personal tillgänglig t.ex.) går det att avbryta efter denna fas utan att särskilt stora resurser har använts.

Under elaborationsfasen utvecklas den övergripande designen och arkitekturen för projektet. Till skillnad från under vattenfallsmodellen uppmuntras att det här skrivs kod för att bevisa att den föreslagna arkitekturen är möjlig och realistisk att implementera. En stor skillnad mellan RUP och Waterfall är att planeringen under RUP inte är lika intensiv vid starten. Under RUP görs en övergripande men inte särskilt detaljerad plan upp över projektets olika faser, sedan görs en mer detaljerad plan upp över den nuvarande samt den nästkommande iterationen. På det här sättet, menar man, undviker man hamna i BDUF-fällan (Big design up front) där allt för stor energi läggs på den inledande kravspecifikationen, krav som ändå kommer att ändras längre fram i processen.

---

[12] Gibbs, *Project Management with the IBM® Rational Unified Process®: Lessons from the Trenches*, Kapitel 2.2
[13] Gibbs, *Project Management… ,* Kapitel 2.2
14 Sinan, “Understanding the Unified Process”*,* i *Methods & Tools* utgåva “Spring 2002”, 2002,

3.1.5 Argument för och emot RUP

Medan vattenfallsmodellen kan sägas fokusera på en idealiserad process, fokuserar RUP mer på flexibilitet och hur projekt i det verkliga livet lyckas respektive misslyckas. Detta gör processen mer adaptiv och flexibel, men också mycket svårare att lära sig. Det tar tid för ett lag av utvecklare att lära sig att arbeta efter RUP-metoder, tid som i vissa fall skulle ha kunna utnyttjats mer effektivt på ett annat sätt. En annan fara med RUP är att man ser det som en ”one size fits all”-process, där processen inte anpassas efter den individuella situationen: *”Beware of purists and extremists, those who focus on the "letter of the UP" rather than the spirit of the UP."*[15]

RUP är som sagt mycket omfattande, och det är svårt att få grepp om vad det egentligen är. Det gör att folk som pratar om RUP ofta kan mena helt olika saker, fast de använder liknande ord: *RUP is designed to accommodate almost anything. I've seen old-fashioned high-ceremony waterfall fit into RUP, and XP fit into RUP. Indeed, my objection to RUP is that it's too vague.*[16]

3.1.6 Utvärdering av RUP

RUP tillhandahåller alltså en flexibilitet som vattenfallsmodellen inte kommer i närheten av, en flexibilitet som Oddsnews definitivt är i ett behov av. Med en enhetlig och väldefinierad uppdragsgivare hade det vart möjligt att ställa upp ”use-cases” i samarbete med denna. Nu är istället Oddsnews en Internettjänst som är öppen för alla, vilket gör det mycket svårt att på ett organiserat sätt samla in use-cases.

En annan nackdel med RUP är just omfånget. Även om det uppmuntras att processen ska anpassas efter den enskildes behov kräver detta en övergripande förståelse för alla delar av processen. Det tar alltså lång tid att sätta sig in i RUP och att lära sig att använda processen på ett effektivt sätt. Eftersom ett av kriterierna för utvecklingen av Oddsnews.com var att snabbt få en färdig produkt talar alltså detta emot RUP som modell för Oddsnews utvecklingsprocess.

3.1.7 Agile Development och Extreme programmering (XP)

En modell för utvecklingsprocesser som i skrivande stund är på modet bland utvecklare är så kallade agila, eller lättrörliga, metoder. De agila metodernas mål är att maximera flexibiliteten och samtidigt minimera den arbetsbörda som en definierad arbetsprocess med nödvändighet innebär. De centrala idéerna hos agila utvecklingsmetoder sammanfattas i det manifest som en grupp av inflytelserika forskare och utvecklare slog fast vid ett möte i februari 2001 på en skidort i Utah, USA:[17]

> **Manifesto for Agile Software Development**
>
> We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:

---

[15] Ibid., s.16

[16] Martin Fowler, “Interview with Kent Beck and Martin Fowler”, http://www.informit.com/articles/article.asp?p=20972&rl=1, 2006-09-23

[17] Highsmith, J., “*History of the Agile Manifesto*”, http://www.agilemanifesto.org/history.html, 2006-09-25

> **Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more. [18]

Ur detta manifest har det sedan vuxit fram en mängd varianter av agila utvecklingsprocesser, som skiljer sig i detaljer men delar dessa grundläggande antaganden. Exempel på sådana metoder är Crystal, SCRUM, Lean software development, Feature driven development, och Extreme programming (XP). I denna uppsats har vi valt att mer gå in i detalj på just XP, eftersom det tar ”proven industry best practices and turns the knobs up to ‘ten’”[19] För att visa på de kontraster som finns mellan olika utvecklingsfilosofier är därmed XP ett utmärkt val, eftersom skillnaderna då kommer att framträda så tydligt som möjligt.

Till skillnad från RUP, som har anklagats för att vara för vagt, består XP av 12 klart definierade principer, som enligt upphovsmännen alla måste följas för att processen ska fungera på ett optimalt sätt. Detta gör XP mer rigid än andra agila metoder, men också lättare att förstå och förklara för andra. XP utvecklades av Kent Beck, Ward Cunnighamn och Ron Jeffries när de arbetade hos biltillverkaren Chrysler under sent 1990-tal.[20] Kent Beck beskriver XP på detta sätt:

> XP is distinctly post-modern. It's philosophical roots are in complex systems theory. In XP, we are looking to create the "ilities" as emergent properties of the whole team acting according a set of simple rules, not through centralized control. In business terms, this results in a team that's robust, because no one person is a bottleneck, and yet can go fast when the way forward is clear.[21]

I sin renaste form kan XP beskrivas med en uppsättning kriterier (se figur nedan). I denna uppsats finns inte plats för att gå igenom alla dessa steg för steg, istället tas de mest centrala upp. För en komplett introduktion till ämnet rekommenderas någon av Kent Becks böcker, t.ex. ”Extreme programming explained”.

> Planning Game
> Testing
> Pair Programming
> Refactoring
> Simple design
> Collective code ownership
> Continous integration
> On-site customer
> Small releases

---

[18] Manifesto for Agile Software Development, http://www.agilemanifesto.org/, 2006-10-12
[19] Miller, R. (2002), ”Demystifying extreme programming”, http://www-128.ibm.com/developerworks/java/library/j-xp0813/, 2006-10-10
[20] Extreme Programming, http://en.wikipedia.org/wiki/Extreme_Programming, 2006-10-10
[21] Fowler, M, Beck, K. (2001), “Interview with Kent Beck and Martin Fowler”, http://www.informit.com/articles/article.asp?p=20972&rl=1, 2006-09-23

40-hour week
Coding Standards
System metaphor[22]

XP kräver att enhetstester ska skrivas för all kod, och att dessa tester ska skrivas innan någon produktionkod skrivs. På detta sätt undviks att utvecklarna skriver extra kod som egentligen inte behövs, den enklaste möjliga koden som klarar testet är det bästa. Parprogrammering är kanske den mest kontroversiella delen med XP. Enligt denna princip ska all utveckling ske i par tillsammans med en annan utvecklare. På det här sättet uppnås enligt upphovsmännen en högre kvalitet på koden, och den minskning i rader skriven kod per timme som detta medför vägs i hög grad upp av det minskade tiden som går åt till felsökning i ett senare skede.[23]
Med refactoring menas att koden skrivs om kontinuerligt, allt eftersom bättre sätt att lösa problem upptäcks. Detta görs utan att påverka den yttre funktionaliteten och det som uppnås är alltså en bättre och effektivare kod internt. Denna princip drar stor nytta av de automatiska enhetstesterna, eftersom det alltid går snabbt att testa så att omskrivningen inte förstör någon av den ursprungliga funktionaliteten.

3.1.8 Utvärdering av Agila metoder / XP

Eftersom Oddsnews är ett väldigt snabbrörligt projekt, som kräver en så stor flexibilitet som möjligt, var det tidigt uppenbart att agila metoder skulle passa dess utvecklingsprocess. Utvecklingsarbetet hos Oddsnews Ltd har influerats av XP i arbetet, dock utan att anamma alla XPs principer. Särskilt viktiga har principerna med enkel design och kontinuerlig refactoring av designen varit. På det här sättet har man inte infört några onödiga funktioner och har kunnat bibehålla en relativt liten kodbas. Dessutom har enhetstester utformats för vissa delar av koden (se avsnitt 5.2 nedan) vilket på ett märkbart sätt gjorde utvecklingen snabbare och mer effektiv.

3.2 Utvärdering av utvecklingsmetoder

Under utvecklingsprocessen har man på Oddsnews Ltd försökt att inte bli slav under någon metod. Förhållningssättet har varit att det finns guldkorn i alla dessa utvecklingssätt och det gäller att välja ut de som fungerar bra för företaget. Dessutom har man hela tiden försökt att undvika att bli paralyserad av modellerna, genom att snarare se vad i modellerna som kan hjälpa det konkreta utvecklingsarbetet än att försöka anpassa utvecklingsprocessen efter en given modell.

Precis som vi antog i vår hypotes så stämmer Agila metoder bäst överens med de förutsättningar som fanns för utvecklingsarbetet på Oddsnews Ltd. Dock är inte heller de agila metoderna ett färdigt recept på hur utvecklingen ska gå till. Under utvecklingen av Oddsnews har man, som redan nämnts, medvetet valt bort vissa av de delar som ingår i Extreme programming, då kanske främst parprogrammering. Detta gjordes av rent praktiska skäl. Alla personer i utvecklingsteamet är bosatta på olika platser och därför är det inte möjligt att bedriva parprogrammering framför en gemensam dator. För att motverka de svårigheter som en geografiskt splittrad utveckling innebär har man i utvecklingsprocessen i möjligaste mån använt tekniska

[22] Stepanek, G. , *Software project Secrets – Why software projects fail*, Apress 2006
[23] Cockburn och Williams i Stepanek George, 2006, s.78

lösningar för att uppmuntra till kommunikation i teamet. Genom att kontinuerligt använda verktyg som MSN, Subversion och vanlig email har företaget syftat till att upprätthålla en effektiv process utan att utvecklarna är lokaliserade till samma plats. Applikationens arkitektur har även den möjliggjort en sådan utspridd utvecklingsprocess genom den lösa koppling som medvetet skapats mellan olika delar av applikationen. Detta innebär att systemets delar i möjligaste mån är oberoende av varandra, och kommunicerar med varandra genom klart definierade gränssnitt. Uppgiften för utvecklarna vid de fysiska möten då man träffas blir då till stor del att diskutera den övergripande arkitekturen, samt att definiera kommunikationsvägarna i systemet.

4. Tekniska lösningar

När vi nu har kommit så här långt in i uppsatsen kan det vara lämpligt att gå in mer konkret på de tekniska lösningar som driver systemet. Som tidigare nämnts så har vi, genom ett medvetet val av teknik, kunnat skapa en utvecklingsprocess som är anpassad för Oddsnews sätt att arbeta.

Valet av verktyg och ramverk vid utvecklingsprocesser är i hög grad ett subjektivt val som görs av utvecklarna. Ofta finns det flera alternativ att välja mellan som är likvärdiga i de flesta avseenden, vilket gör att det i slutändan är upp till programmerarnas personliga preferenser att göra valet. Där spelar tidigare erfarenheter av verktyget ofta en avgörande roll. Krav kan även komma från till exempel kunden eller från projektledningen. Syftet med det här kapitlet är därmed inte att göra någon vetenskaplig jämförelse mellan de olika verktygen som valts, utan att redogöra för den typ av praktiska resonemang som ligger bakom val av verktyg och ramverk för ett utvecklingsprojekt som detta.

Det första valet som måste göras när ett nytt system ska byggas upp är vilket programmeringsspråk som ska användas. De vanligaste språken för webapplikationer idag är Java och Microsofts .Net, men det finns även andra, t.ex. Python eller Ruby. I vårt fall fanns en tidig prototyp färdig när vi kom in i projektet, och valet av Java som språk var därmed redan taget.

Givet Java har många val redan tagits för oss, men det finns fortfarande en hel del utrymme att röra sig i. Det första valet, efter vilket programmeringsspråk som används, blev vilken utvecklingsmiljö som ska köras. I javavärlden finns det, lite hårddraget, två alternativ för så kallade IDE:s (Integrated Development Enviroment), Eclipse samt Netbeans. Vilken av dessa man väljer beror lite på vilka behov man har, Eclipse har i allmänhet fler tillägg, så kallade plug-ins och därmed en större flexibilitet, medan Netbeans enligt många är smidigare att använda. Valet av utvecklingsmiljö för en webapplikation är därmed till stor del en fråga om vad utvecklarna har använt tidigare och vad de känner sig bekväma med, i Oddsnews fall föll valet på Eclipse. Skulle vi däremot ha skrivit ett vanligt skrivbordsprogram är förmodligen Netbeans att föredra, eftersom den innehåller en väldigt bra användargränssnitt-byggare.

Ett annat val som måste göras tidigt i processen är vilka eventuella så kallade ramverk som ska användas. Ett ramverk kan enkelt sägas vara en samling färdiga funktioner som programmerarna kan bygga vidare på, för att på så sätt slippa ”uppfinna hjulet” varje gång ett system ska utvecklas. Vid utveckling av webapplikationer är det vanligt att använda ett så kallat MVC-ramverk (Model-view-controller), för att få en klar avgränsning och så stor självständighet som möjligt mellan de olika delarna i systemet. De två klart populäraste webramverken för Java är Struts och Spring. Struts är äldre än Spring, och används av väldigt många system världen över idag. Att valet för Oddsnews föll på Spring berodde till viss del på att vi ville lära oss en ny teknik, samt att Spring, förutom MVC-delen, tillhandahåller så kallat IoC-stöd (Inversion of control, mer om det senare). Spring hjälper även till att använda Hibernate på ett effektivt sätt, som vi kommer till i nästa stycke.

Oddsnews affärsidé bygger på att samla in stora mängder information och presentera detta på ett överskådligt sätt. Därför måste vi ha något sätt att lagra denna mängd av information. Ett naturligt val när vi pratar om stora datamängder är att ha en databas i botten. Det mest populära databasservern idag är förmodligen MySQL[24] men det finns också andra alternativ, för Oddsnews var dock MySQL det naturliga valet. Förutom MySQL behövdes även en komponent som ”översätter” informationen mellan den objektorienterade form i vilken data representeras i Java, och den tabellbaserade form i vilken den lagras i relationsdatabasen. För användningsområden som detta är ramverket Hibernate standard i javavärlden. Senare i det här kapitlet går vi in mer ingående på hur denna översättning går till.

När vi nu i detta avsnitt kortfattat presenterat de tekniska lösningarna kan det kanske tyckas att alla dessa komponenter måste kosta en massa pengar. Men det som gör Java-världen unik är att utbredningen av så kallade Open Source-komponenter, d.v.s. komponenter med fri källkod, är mycket stor. Det går alltså att bygga en hel webapplikation med mycket hög kvalitet och skalbarhet enbart med hjälp av fria komponenter. Detta var givetvis även det en avgörande anledning till att Oddsnews valde att använda Java, eftersom en stor begränsning i processen har varit bristen på kapital.

I nästa avsnitt kommer vi att gå in på det som ligger i botten på applikationen, nämligen Servermjukvaran, och på vilket sätt den har behövt anpassas och konfigureras för Oddsnews behov.

4.1 Server

Ett viktigt val under utvecklingen av Oddsnews var på vilken sorts server applikationen skulle köras. Till stor del är det valet av programspråk för utvecklingen som styr vilka alternativ som finns möjliga, men just för Java / Jsp finns det ett otal olika alternativ. Detta beror på att Servlet-teknologin är en öppen standard. En organisation, Java Community Process, ger ut så kallade JSR:s (Java specification requests, jämför med RFC) som anger hur en produkt ska fungera för att uppfylla en viss specifikation. Tekniskt sätt så är en JSR en samling med interface, samt en beskrivning av vad metoder i dessa interface ska utföra. Sedan är det upp till de företag som utvecklar programvara baserad på en JSR att avgöra hur detta ska implementeras.[25]

Ett annat val som behövdes göras var om webapplikationen skulle köras på en fullfjädrad applikationsserver, eller om det räckte med en ”servlet container”. Eftersom Oddsnews inte använder de extra funktioner som JEE (Java Enterprise Edition) och en fullskalig applikationsserver för med sig togs beslutet att istället köra med Tomcat / Apache, ett utförligare resonemang om detta kommer på nästföljande sidor.

4.1.1 Val av operativsystem

I början kördes servern på en gammal dator med Windows 2000. Eftersom webbplatsen då endast var tillgänglig för oss utvecklare och ett litet antal vänner och bekanta var detta tillräckligt.

---

[24] MySQL är idag tveklöst en av världens mest kända databasservrar, om inte den mest kända, och huvudkontoret ligger i Uppsala.
[25] “Sun Java Community Process Program”, http://jcp.org/en/home/index, 2006-10-13

Allteftersom utvecklingsarbetet fortgick uppfattades det dock som mer kostnadseffektivt och mer driftsäkert att köra en Linux-version på servern, varför vi installerade och konfigurerade en sådan server. Eftersom en av författarna till arbetet var bekant med Ubuntu Linux valdes den som distribution.

Efter en utvärdering av våra alternativ bestämde vi oss för att köpa en IBM-server med en 2.8Ghz processor och 1gb RAM-minne. Som ett nystartat företag utan riskkapital var ekonomin självklart mycket begränsad, men servern ansågs som tillräcklig för de krav som fanns i ett inledande skede. Däremot var ett viktigt krav att man skulle kunna distribuera systemet till fler servrar när behov uppstod. Detta krav fick stor betydelse i valet av mjukvara på servern.

4.1.2 Val av servermjukvara

Den mest kända ”servlet containern” är Apache Tomcat, det var också den vi använde i början. Vi använder även Tomcat på våra utvecklingsdatorer, eftersom det är lätt att konfigurera och det finns ett bra stöd för Tomcat inifrån utvecklingsmiljön Eclipse.

När vi senare skulle starta upp på riktigt undersöktes även andra alternativ för att se vilket som passade bäst till Oddsnews. Ett alternativ som tidigt dök upp var att använda Apache tillsammans med mod_jk, ett tillägg till Apache som gör det möjligt att använda Tomcat som en container till Apache. Mod_jk ger samtidigt möjlighet att använda Apache för att serva det statiska materialet som t.ex. bilder, JavaScript och css-filer, medan de dynamiskt genererade sidorna skapas utav Tomcat-servern. Traditionellt sätt har detta setts som en stor fördel, eftersom Apache varit snabbare än Tomcat, men prestandagapet mellan dessa har minskat för varje år, och är idag inte längre någon avgörande faktor vid serverval.[26]

Den stora fördelen med att använda Apache som server är istället den mängd av tillägg som finns till denna. På Oddsnews bestämde vi oss efter noggrant övervägande för att använda ett antal, av vilka vi kommer ta upp fyra här: mod_jk, mod_deflate, mod_expires, och mod_svn.

Eftersom vår applikation använder mycket JavaScript (se avsnittet om Ajax) så krävs det en relativt stor nedladdning första gången en användare besöker sidan. Därför undersökts tidigt på vilka sätt man kunde snabba upp den nedladdningen och göra den så smidig som möjlig för användaren. Det första vi gjorde var att rensa i javascripten, genom att ta bort kommentarer och snygga upp koden minskades storleken drastiskt. Två tillägg till Apache gjorde sedan en stor skillnad i den upplevda snabbheten för användaren, mod_deflate och mod_expires.

4.1.2.1 Mod_deflate

Den begränsande faktorn, flaskhalsen, vid webtrafik är nästan alltid bandbredden. Mod_deflate bygger på den enkla ideén att materialet komprimeras på serversidan och skickas sedan till klienten. Klientens webläsare packar sedan upp det mottagna datan, och visar den. Detta sker helt transparent för användaren, denne märker alltså ingenting, förutom att websurfandet förmodligen känns smidigare.

---

[26] *Tomcat FAQ, ”*Is Tomcat faster than serving static HTML pages than apache?*”* http://tomcat.apache.org/faq/performance.html, 2006-10-10

Genom att använda mod_deflate så komprimeras Oddsnews förstasida från runt 50kb, till 8kb, alltså en komprimeringsgrad på mer än 80%. På samma sätt är det för javascripten och det övriga textinnehållet, komprimeringsgraden ligger generellt mellan 75-85%. Dock så skulle mod_deflate inte göra någon större skillnad på bilder, som ofta redan är komprimerade, därför exkluderas dessa från komprimeringen.

Nackdelen med mod_deflate är att det krävs processorkraft hos både server och klient för att komprimera respektive dekomprimera filerna. Detta ansågs dock vara en väldigt liten faktor, eftersom just processorkraft inte är en bristvara på servern, och oftast inte heller hos klienten. Att materialet komprimeras innan det skickas iväg gör också att servern behöver ägna mindre processortid åt varje klient, eftersom överföringen går så pass mycket snabbare.

Det finns stöd för gzip-komprimering av data i alla de stora webläsarna från version 4 och framåt. Om en webläsare stöder komprimerat material skickar den en ”Accept-encoding:gzip” i request-headern, på det sättet kan servern avgöra om den ska komprimera materialet eller inte. Mod_deflate är alltså helt bakåtkompatibelt med webläsare som inte stödjer komprimering av datan från servern.

4.1.2.2 Mod_expires:

Ett annat sätt att snabba upp surfandet för klienten är att sätta expires-headern i svaret från servern. Genom att göra detta så fås klientens webläsare att ladda filen från den lokala cachen istället för att ladda från servern, om inte klockslaget i expiresheadern har passerats. Eftersom sidorna på Oddsnews är dynamiska så bör de inte cachas någon längre tid, därför sattes utgångstiden för html-sidor till tio minuter. Däremot så förändras bilder och JavaScript inte lika ofta, därför kunde utgångstiden göras längre för dessa. När sidan är i färdigt och stabilt tillstånd bör utgångstiden kunna göras ännu längre för sådant material, eftersom de sällan förändras oftare än en gång per dag.

Mod_expires är precis som mod_deflate transparent för användaren, utom i ett fall: Då css:er eller JavaScript precis uppdaterats på servern och de ligger kvar i en användares cache. Detta yttrar sig då ofta genom att layouten ser konstig ut, eller att någon funktion som beror på JavaScript inte fungerar. Dock är detta inget större problem, eftersom materialet bara cachas i 24 timmar hos klienten, tills dennes nästa besök är alltså problemet förmodligen löst. Besökaren kan också ladda om sidan genom att klicka på ”refresh”-knappen, då laddas de nya filerna ner från servern.

Denna nackdel ansågs som så liten att det var värt att ändå använda Mod_Expires, eftersom det ger en stor minskning av använd bandbredd.

4.1.3 Distribuering

Den absolut viktigaste faktorn när servermjukvara skulle väljas var möjligheten att växa med ökande trafik. Det krävdes alltså någon form av distribueringsmöjlighet, det vill säga möjligheten att dela arbetet mellan flera olika servrar. Tomcat har som standard ingen möjlighet att dela arbetet mellan flera olika datorer. Det är i och för sig möjligt att lägga in flera inlägg för domännamnet oddsnews.com i dns-servrarna så att klienterna genom den vägen skickas till olika servrar. Varje gång en klient frågar efter IP-adressen till oddsnews.com lämnar då DNS-servern IP-numret till en av de olika webbservrarna

enligt ett ”Round-Robin”-schema, det vill säga en enkel turordning.[27] Nackdelen med detta är att det inte lämnar någon kontroll till oss som ägare att avgöra vilken trafik som ska skickas var, vilket gör att det inte är möjligt att anpassa fördelningen efter skillnader i olika servrars prestanda, därför var det mycket viktigt för oss att lösa detta problem. Det är även ett problem eftersom klienten, efter att ha besökt webbplatsen första gången, cachar DNS-uppslagningen lokalt på den egna datorn för att snabba upp senare besök, detta gör att trafikdelningen inte fungerar som det är tänkt.

När Apache körs med mod_jk krävs det en hel del konfiguration som inte behövs när Tomcat körs ensam, dock ger detta en större flexibilitet samt möjligheten att använda tillägg, som redogjorts för tidigare.

Genom att definiera ett antal olika ”workers” i en konfigurationsfil för Apache kan mod_jk fås att distribuera trafiken till ett godtyckligt antal Tomcatprocesser, som kan köras på olika datorer. Det går även att sätta olika vikt på olika servrar, så att trafikmängden de får kan anpassas efter eventuella skillnader i hårdvara. Det löser därmed problemet som redogörs för i stycket ovan. I början definierades en ”worker”, som kördes på samma maskin som Apache-processen. När vi sedan lägger till flera datorer kan en worker (eller flera) köras på samma dator, och kommunicera med Apacheservern.

Nackdelen med detta är självklart att datorn som kör Apache blir en ”single point of failure”, om den inte fungerar så fungerar heller inte sidan. För riktigt stora webplatser behöver man göra något åt detta (kanske kombinera med DNS-tekniken som redogjordes för ovan), men för Oddsnews så räcker detta än så länge långt.

Det enklaste sättet att dela arbetet mellan flera datorer är förmodligen att flytta MySQL-servern till en egen dator. När trafiken sedan blir större än vad en server kan orka med finns det även flera lösningar för att distribuera databaser, t.ex. genom att lägga olika tabeller på olika datorer, men det valdes tidigt bort i avgränsningen av det här arbetet, eftersom det inte sågs som aktuellt på ännu ett långt tag.

Det bästa sättet att distribuera arbetet när sajten fortfarande är relativt liten vore att låta webservern ligga på en dator, MySQL-servern på en annan och extraktorerna på en tredje. På det sättet skulle man enkelt kunna uppdatera extraktorerna utan att, som vi var tvungna i början, att ta ner hela webapplikationen. Eftersom weblagret och extraktorerna aldrig skriver till samma tabeller i databasen så undviks helt konsistensproblem med denna lösning, något som annars kan vara ett stort problem i distribuerade arkitekturer.

4.2 Övergripande arkitektur

Eftersom vi hela tiden har arbetat under tidspress under utvecklingen av Oddsnews vill man så långt som möjligt bygga applikationen på färdiga lösningar. Vi ville också uppnå så lösa kopplingar och så bra separation som möjligt mellan olika delar av

[27] Couloris, G. et.al, *”Distributed Systems – Concepts and Design”,* 3rd Edition, Addison-Wesley, Pearson Education, England, 2001, s.369-370

systemet. Detta gör det möjligt att arbeta utspritt utan daglig kontakt mellan olika utvecklare, vilket var en omöjlighet för Oddsnews av praktiska skäl. För att uppnå detta valdes Spring ut som ramverk, dels för dess MVC-del (Model-view-Controller), och dels för dess stöd för DI (Dependency Injection, också kallat IoC).

4.2.1 Spring

Spring är ett ramverk för J2EE-utveckling som har uppnått en stor popularitet över de senaste åren. Trots att det kanske främst används för webbutveckling har det många användningsområden utöver detta.

En bra utgångspunkt för diskussionen om Spring kan vara att utgå från några av de fördelar med Spring som dess huvudutvecklare Rod Johnson framhåller i artikeln "Introducing the Spring Framework"[28]:

- Spring kan begränsa användningen av Singletons i ett projekt.
- Spring kan eliminera användningen av flera olika filformat för konfigurering av applikationen genom att tillhandahålla ett konsistent format för hela projektet.
- Spring uppmuntrar god programmeringssed genom att göra det väldigt enkelt att programmera mot interface.
- Spring är designat så att applikationer som byggs med beror på så få av dess API:s som möjligt. De flesta affärsobjekt i Spring har inget beroende av just Spring, vilket gör att inlärningströskeln blir låg. Det blir också lätt att återvinna delar av applikationen om man i framtiden skulle vilja byta ramverk.
- Applikationer byggda med Spring är enkla att enhetstesta.

Den kanske viktigaste och mest spännande delen av Spring är dess användning av så kallad "Dependency Injection" och "Inversion of Control". Spring är inte på något sätt ensamt om att använda dessa metoder, ett exempel på ett alternativ är Picocontainer[29]. Picocontainer är dock inget ramverk på samma sätt som Spring, utan har som mål att enbart vara en "behållare" (container) för affärsobjekt.

Inversion of Control kan kort sägas vara när en utomstående enhet (Spring-containern i vårt fall) tillfredställer objektens beroenden, istället för att programmeraren själv måste göra detta. När man som programmerare börjar använda Spring är det lätt att få intrycket att ramverket inte gör någonting, eftersom de flesta objekt är POJO:s[30] som själva inte beror på Spring. Men det är just här styrkan med Spring och inversion of control ses tydligast: genom att låta objekten i så stor mån som möjligt vara oberoende av den container de använder så går det lätt att t.ex. byta till ett annat framework om behovet skulle uppstå i framtiden. Inlärningsperioden blir också kortare eftersom programmeraren kan skriva vanliga POJO:s och inte behöver lära sig ett nytt API för att kunna använda Spring.

---

[28] Johnson, R., "*Introduction to the Spring Framework",*
http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework, 2006-08-20

[29] http://www.picocontainer.org

[30] Plain Old Java Objects, alltså ett helt vanligt javaobjekt utan beroenden på externa ramverk. Begreppet myntades 2000 av Martin Fowler, Rebecca Parsons och Josh McKenzie: "We wondered why people were so against using regular objects in their systems and concluded that it was because simple objects lacked a fancy name. So we gave them one, and it's caught on very nicely". (http://en.wikipedia.org/wiki/Plain_Old_Java_Object, 2006-10-13)

4.2.2 Design patterns

Ett design pattern kan sägas vara en återanvändningbar lösning på ett återkommande problem inom objektorienterad design. Christopher Alexander beskriver design patterns på följande sätt, och även om han egentligen skriver om arkitektur är likheterna med mjukvaruutveckling slående:

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice"[31]

Design Patterns har varit en del av utvecklarnas gemensamma kultur sedan begynnelsen, men det var först 1994, när de så kallade ”Gang Of Four”[32] släppte sin inflytelserika bok ”Design Patterns – Elements of reusable Object-oriented software”, som de blev ”standardiserade”, så att olika utvecklare kan prata med varandra om ett pattern och vara säkra på att mena samma sak.

För en snabb och effektiv utvecklingsprocess är det självklart att man ska försöka bygga vidare på det andra tidigare gjort, och därför ville vi i så stor utsträckning som möjligt använda patterns i vår kod. Patterns som Factory, Builder och Singleton blev däremot till stor del överflödiga genom vår användning av Spring som framework. Det dessa Patterns har gemensamt är att de behandlar skapande och åtkomst av objekt, och genom att använda Spring på ett smart sätt kan ramverket fås att hantera livscykeln för affärsobjekten.

Ett ofta använt design pattern i JEE-applikationer är DAO, Data Access Object. Det har till uppgift att fungera som ett mellanlager mellan databasen och affärslogiken. På det sättet uppnås en separation (”separation of concerns”) mellan deras respektive ansvarsområden så att koden inte flyter in i varandra, vilket är önskvärt för att koden ska bli så lätt som möjligt att underhålla och bygga ut.

> The primary purpose of the DAO pattern is to separate persistence-related issues from general business rules and workflows. We don't want the business or workflow logic to depend on or have any knowledge of what data access technology is actually used.[33]

Spring tillhandahåller ett antal abstrakta DAO-basklasser för olika bakomliggande databaslösningar. Dessa kan sedan implementeras efter projektets behov.

Används Hibernate utan Spring används vanligen en *SessionFactory* för att skapa sessioner som utvecklaren själv hanterar livscykeln för. När man använder Springs DAO-support slipper man göra detta, ramverket tar då över skapandet och underhållandet av sessioner. [34]

---

[31] Alexander, C. i Gamma, et.al, ”*Design Patterns – Elements of reusable Object-oriented software”*, 1994, s.12
[32] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
[33] Johnson R et.al.(2005) *”Proffessional Java development with the Spring framework*”, John Wiley & Sons, Kap 5.
[34] Ibid, Kap.7, avsnitt ”Hibernate”

4.2.3 Spring i praktiken

Ett exempel från Oddsnews kan underlätta förståelsen för IoC och DAO:er: varje oddskälla implementerar interfacet OddsSource, som enbart definierar en metod:

```
List<OddsInfo> getOddsInfo();
```

Denna metod returnerar en lista med OddsInfo-objekt, som innehåller oddsen för de matcher som varje spelbolag har. Denna parsning ser helt olika ut för olika spelbolag, men eftersom alla implementerar samma interface går alla dessa att utnyttja på samma sätt.

Den klass som använder alla oddsextraktorer heter OddsUpdateExecutor. Den körs med jämna mellanrum (för närvarande varannan timme) och lagrar (eller uppdaterar) oddsen i databasen. För att lagra oddsen i databasen har den hjälp av en DAO[35], mer specifikt EventDAO, som definierar metoder för att lagra och ta ut matcher ur databasen. EventDAO:n är en singleton, alla klasser i applikationen använder alltså samma objekt, både när oddsen lagras och när de plockas ut till view-lagret. Skulle vi göra detta utan hjälp av Spring skulle vi på något sätt behöva göra EventDAO:n tillgänglig för alla klasser. Med Spring kan vi däremot enbart definiera enkla getter/setter-metoder i OddsUpdateExexcutor, samt lägga till några rader i konfigurationsfilen, så "injekteras" DAO:n till OddsUpdateExexcutorn automatiskt.

I OddsUpdateExecutor.java:

```
private EventDAO eventDAO;
.
.
public EventDAO getEventDAO() {
     return eventDAO;
}

public void setEventDAO(EventDAO eventDAO) {
     this.eventDAO = eventDAO;
}
```

I applicationContext.xml:

```
<bean id="oddsUpdate"
          class="com.oddsnews.odds.OddsUpdateExecutor">
.
. [Definitioner av alla oddsextraktorer]
.
<property name="eventDAO" ref="eventDAO" />
</bean>
```

Vill vi sedan använda DAO:n i andra klasser är det bara att lägga till enkla getter/setter-metoder i klassen, samt lägga in en <property name="eventDAO" ref="eventDAO"/> i xml-filen. Genom att använda den här typen av injektion blir det väldigt lätt att se vilka beroenden en klass har samt hur den har konfigurerats, all sådan information finns i xml-filen.

[35] Data Access Object

Genom att Spring tar hand om de beroenden klassen har blir det lätt att koda. En annan fördel med detta är att EventDAO är ett interface, som i sin tur implementeras av EventDAOHibernate. Som vi ser är det enda OddsUpdateExecutor vet är att det har ett objekt som implementerar EventDAO, skulle vi i framtiden byta till ett annat O/R-lager går det alltså mycket lätt att byta detta. Genom att använda Spring slipper vi binda oss till en speciell implementation direkt i javakoden, vi slipper alltså skriva EventDAO eventDAO=new EventDAOHibernate() och hantera livscykeln för detta objekt själva i javakoden. Applikationen kan alltså konfigureras att använda olika implementationer av ett interface enbart genom att ändra i xml-filen, vilket är en stor fördel för applikationer där utbyggbarhet och modularitet är viktigt.

4.2.4 Hibernate

Hela idén med Oddsnews bygger på att stora mängder information kan samlas in och behandlas på ett effektivt sätt. Just detta gjorde att en databaslösning var det enda rimliga alternativet. Java tillhandahåller ett API, för åtkomst för SQL-databaser. JDBC[36], Java Database Connectivity, ger ett enhetligt interface för alla databaser, oavsett vilken det är, givet att det finns en drivrutin skrivet för den aktuella databasen.

Alla som har arbetat med JDBC under en tid vet dock att det kan vara krångligt att arbeta med. Även skickliga programmerare har ofta problem med att hantera JDBC och det som krävs, vilket kan leda till buggar som är svåra att hitta och åtgärda.

Det är här så kallade ORM[37]-lösningar kommer in i bilden. En relationsdatabas lagrar sin information i tabeller, där olika tabeller kan vara relaterade till varandra på olika sätt. I ett objektorienterat programmeringsspråk som Java har också objekt relationer med varandra, men de går inte direkt att lagra i en databas. Därför finns lösningar som ”översätter” de objektorienterade datastrukturerna till ett lämpligt format för att lagra i databasen.

Hibernates uppgift är alltså att ”mappa” de datastrukturer som definierats i javakod till motsvarande databaschema och tabeller i SQL-databasen. Genom att arbeta med Hibernate förkortades utvecklingstiden av Oddsnews betydligt, samtidigt som antalet buggar förmodligen blev färre.

Vi valde också att använda Hibernate Annotations för att specificera hur OR-mappingen ska gå till. Alla OR-lösningar kräver att användaren specificerar meta-data för hur och vad som ska lagras i databasen. Innan Java 1.5 och Hibernate Annotations skrevs dessa specifikationer i speciella xml-filer, eller med ett hjälpramverk som t.ex. XDoclet[38], men med annotations kan de istället skrivas direkt i javakoden. Ett exempel på hur detta gjordes visas nedan:

```
@Entity
@AccessType("field")
public class TrottingEvent extends AbstractEvent{
     @OneToMany(cascade = { CascadeType.ALL }, fetch = FetchType.EAGER)
     @JoinColumn(name = "trottingEvent_fk")
```

[36] Java SE Techonologies, Java Database Connectitivity (JDBC)
http://java.sun.com/javase/technologies/database.jsp , 2006-10-10
[37] Object / Relational Mapping
[38] Xdoclet, http://xdoclet.sourceforge.net/

```
    private List <Competitor> competitors = new
ArrayList<Competitor>();

    @Enumerated(EnumType.STRING)
    @Column
    private TrottingLocation location;

    @Column
    private int raceNumber=-1;
```

```
@Entity
public class Competitor {

    @Id
    @GeneratedValue
    protected Long id;

    @Column
    private String name;

    @CollectionOfElements(fetch= FetchType.EAGER)
    private List <SingleOdds> singleOdds= new ArrayList<SingleOdds>();
```

Av störst intresse här är kanske de olika sätten att modellera relationer. Hibernate stöder modellering av både en-till-en, en-till-många och många-till-många-relationer. Med @CollectionOfElements kan man definiera relationer mellan en ägarklass och en lista av vilken datatyp som helst, även primitiva typer som int och double.[39]

Med @OneToMany i TrottingEvent-klassen definierar vi att ett travlopp kan ha många deltagare. En relation i en relationsdatabas är alltid dubbelsidig, d.v.s. det går alltid att från ett travlopp hitta dess deltagare, men det går också att, utifrån en deltagare, ta reda på vilket lopp denne deltar i. I Java däremot så kan (och är ofta) en relation enkelsidig, d.v.s. att klassen TrottingEvent har referenser till de Competitors som deltar i loppet, däremot har en Competitor ingen vetskap om vilket lopp denne deltar i. Detta var ett medvetet designval eftersom vi aldrig såg ett behov av att plocka ut en tävlande direkt, utan åtkomsten av dessa går alltid genom att vi plockar ut ett visst travlopp ur databasen.

Hade vi dock velat kunna gå åt båda hållen hade det varit möjligt med annotationen @ManyToOne som visas i exemplet nedan. Genom att använda detta hade varje Competitor fått en referens till ”sitt” lopp som denne tillhör.

```
@ManyToOne
@JoinColumn(name = "trottingEvent_fk", insertable = false, updatable =
false)
    TrottingEvent event;
```

[39] Hibernate Annotations – Reference Guide, http://www.hibernate.org/hib_docs/annotations/reference/en/html_single/#entity-hibspec-collection, 2006-09-20

För att Hibernate ska kunna hålla reda på vilket objekt som hör till vilken rad i databasen används en unik identifierare för varje instans av klassen som ska sparas. Genom att använda annotationerna @Id och @GeneratedValue så skapar Hibernate själv ett Id för varje klass utan att vi som programmerare behöver göra någonting. För att spara eller uppdatera ett objekt behövs sedan bara en rad kod:

```
trottingDAO.makePersistent(event);
```

4.3 Extraktorer

Den centrala strategin för Oddsnews.com är alltså att samla in information och att sammanställa den på ett unikt sätt. Detta görs genom att skapa javaklasser som mellan angivna intervall hämtar in speciell information från diverse källor. Dessa klasser kommer nedan att kallas för extraktorer eller parsers – två begrepp som kommer att användas som synonymer.

Den centrala idén bakom dessa extraktorer är att arbeta igenom en textfil på en viss url och hämta ut den informationen i den som är relevant. Sättet detta görs varierar dock mycket beroende på hur textfilen är strukturerad. I vissa fall kan filen redan vara strukturerad på ett sätt som gör det enkelt att extrahera informationen ur den. Exempel på detta kan vara xml-filer som andra webbtjänster tillhandahåller för att de vill att deras information ska reproduceras. Här kan man använda så kallade *xml-parsers*. I andra fall kan man vara tvungen att hämta in information som inte är strukturerad på sådant sätt – exempelvis direkt från en html-sida. I dessa fall blir man istället tvungen att leta efter speciella mönster i den textsträng filen innehåller. För detta ändamål finns det ett standardiserat språk där man kan beskriva hur dessa mönster ska se ut – *Regular expressions*.

4.3.1 Regexp-parsers

Som nämnts ovan används *regular expressions* för att beskriva hur man ska hämta ut information ur återkommande mönster i textsträngar. Detta görs genom att definiera en textsträng bestående av tecken som har antingen normal eller en speciell semantisk betydelse. I sin enklaste form består *regular expressions* bara av en vanlig söksträng. Exempelvis kommer strängen ”Pelle” hitta varje förekomst av det ordet i en längre text. Detta är dock inte vad som gör regular expressions unikt gentemot tidigare implementationer av sökalgoritmer. Genom att använda så kallade speciella tecken kan man nämligen få regular expression att känns igen betydligt mer komplexa mönster. Den ovannämnda strängen ”Pelle” kommer i praktiken bara ge tillbaka de index i strängen vi söker i där mönstret ”Pelle” finns. Sätter vi däremot strängen inom parantes, ”(Pelle)” – kommer Pelle att räknas som en grupp – vilket innebär att vi även får tillbaka just strängen Pelle varje gång vi hittar den. Kanske inte det mest användbara – men i kombination med andra tecken blir detta, vilket ni snart kommer se, väldigt användbart. Skriver vi nämligen vårt *regular expression* som ” (.{*}) ” kommer vi att hitta en grupp med godtyckliga tecken och godtycklig längd – så länge den är innesluten av ett och ett . Denna grupp skulle alltså exempelvis kunna vara Pelle, Kalle, eller vilket annat namn som helst. Vidare kan gruppen definieras i olika strikta termer. Vill vi inte inkludera godtyckliga tecken, eller hämta en grupp av valfri längd, kan detta definieras. Exempelvis kan vi definiera en grupp som ”(\d{2})” – vilket kommer att hitta alla förekomster av två siffertecken som står bredvid varandra.

Dessa *regular expressions* är mycket effektiva vid informationsutvinning på Internet. Utan större arbetsinsats kan man strukturera relativt ostrukturerade informationsmönster – där kanske webbsidor som snarare är skrivna för en mänsklig mottagare snarare än en maskinell kanske är det tydligaste exemplet. En viktig bidragande anledning till att detta tillvägagångssätt är effektivt är att informationen på webbsidorna ofta genereras automatiskt utifrån databaser. I och med detta får sidorna oundvikligt en repetitiv karaktär vilket är en viktig förutsättning för att *regular expressions* ska fungera bra. En tydlig fördel här är alltså att man rent tekniskt kan inhämta information från en avsändare utan att den är medveten om att man gör det. I praktiken blir dock den etiska och juridiska problematik vi nämnt ovan som mest påtagligt i dessa fall. Här får man alltså i varje enskilt fall avgöra om motparten vill att man ska mångfaldiga deras information och ifall man har rätt att göra det.

En tydlig nackdel med detta sätt att inhämta information med hjälp av *regular expressions* är att det lägger över ansträngningen för struktureringen av informationen på mottagaren snarare än avsändaren. Ändrar avsändaren exempelvis sin webbsida tillräckligt mycket kan man som informationsinsamlare vara tvungen att delvis eller helt skriva om sitt *regular expression*. Om man hämtar in information från en stor mängd föränderlig information kan detta till slut bli en stor uppgift och i sådana fall kan det vara en fördel om avsändaren har en möjlighet att leverera informationen till en i ett oföränderligt och strukturerat format.

4.3.2 XML-parsers

I många fall av informationsinsamlande är det tillvägagångssätt som beskrivs ovan onödigt komplicerat. Det tydligaste exemplet på detta är när avsändaren redan har strukturerat sin information på ett sätt som enkelt för en dator att läsa. I Oddsnews Ltd fall gäller detta exempelvis när spelbolagen själva har satt ihop en så kallad XML-feed med de odds de har – som de gör tillgänglig för sina samarbetspartners. Även om strukturen på dessa XML-feedar förändras ibland är de överlag ofta betydligt stabilare än webbsidorna där samma information presenteras. Dessutom är informationen presenterad i hierarkisk form så att det är mycket enkelt att skriva en algoritm som på ett effektivt sätt extraherar den information man vill ha ur den (se figur 1).

Det finns många fördelar med att få informationen i XML-format. Ur ett tekniskt perspektiv är den tydligaste att informationen är oberoende av presentationsform. Till skillnad från när man hämtar informationen direkt från webbsidor kommer XML-feeden inte att förändras om avsändaren exempelvis väljer att göra om layouten på sin webbtjänst. Ytterligare en icketeknisk fördel är att man när man hämtar informationen från XML-feeden kan vara relativt säker på att avsändaren vill att informationen ska reproduceras. Ofta står informationens tillåtna användningsområde definierat i någon form av avtal – vilket förenklar de etiska och juridiska frågeställningarna kring vilken information man får extrahera avsevärt.

En XML-extraktor kan enkelt vandra stegvist genom den hierarkiska strukturen i XML-filen och plocka ut den information som är intressant. Utifrån exemplet i figur 2 skulle algoritmen identifiera varje matchnod och för varje sådan hämta ut datumparametern, hemmalaget och bortalaget i noderna under <teams> och sedan hämta ut oddsen i noderna under <odds>. Något som delvis komplicerar extraherandet för Oddsnews Ltd är att det inte finns någon standardiserad mall enligt spelbolagen strukturerar sina xml-

feedar, vilket innebär att man är tvungen att skriva separata javaklasser för varje spelbolag för att hantera de relativt stora skillnader i struktur mellan de olika feedarna.

```
<spelbolag>
     <match date="2006-09-20 19:00">
          <teams>
               <home>AIK</home>
               <away>DIF</away>
          </teams>
          <odds>
               <home>1.5</home>
               <draw>1.8</draw>
               <away>1.7</away>
          </odds>
     </match>
</spelbolag>
```

Figur 2 - Exempel på XML-feed.

4.3.3 RSS

RSS är en standard som används för att syndikera webbinnehåll[40]. Förkortningen kan stå för en av tre standarder, Rich Site Summary (RSS 0.9x), RDF Site Summary (RSS 0.9 och 1.0) samt Really Simple Syndication (RSS 2.x). RSS används huvudsakligen för att förmedla nyheter på ett sätt som imiterar push-teknik. RSS finns just för att undvika de problem med olika struktur som nämndes ovan om XML-feedar. Om en webbsida publicerar sin information som en rss-feed kan denna information inhämtas av en standardiserad rss-läsare som inte behöver anpassas för att förstå strukturen hos avsändaren. Tjänsten Oddsnews.com använder detta för att hämta in ett stort antal nyhetsingresser från flertalet av de stora svenska nyhetsmedierna – något som i praktiken skulle vara mycket komplicerat om de olika nyhetstjänsterna använde egna XML-strukturer så som spelbolagen gör. RSS-feedar kan alltså ses som en motsats till informationsinhämtning direkt från html-filer. De ställer mycket låga krav på strukturering hos mottagaren men däremot relativt höga hos avsändaren.

Ett problem med nyhets-RSS-feedar är dock att de inte på samma sätt som spelbolagens feedar har som syfte att leverera information som ska reproduceras på andra sidor. Det huvudsakliga syftet med nyhetstjänsternas rss-feedar är istället att möjliggöra för deras läsare att på ett smidigt sätt kunna tillgängliggöra sig informationen utan att behöva surfa in på deras webbtjänst. Att informationen samtidigt kan inhämtas av automatiska läsare är bara en bieffekt. På detta sätt liknar insamlandet av informationen i rss-feedarna i etisk och juridisk mening kanske mer det sätt man insamlar information direkt från webbtjänster med *regular expressions* snarare än när man samlar in spelbolagens information från deras XML-feedar. Visserligen finns det i Sverige en citaträtt – vilken tillåter att man refererar delar av någons verk om man samtidigt refererar till det ursprungliga verket (i detta fall i form av en URL). Dock är det mer osäkert om citaträtten gäller om man på ett systematiskt sätt inhämtar information utan att tillföra egen information eller tolkning av den inhämtade informationen.

I praktiken är dock dessa juridiska problem inte så stora som de kan verka. Redan idag finns det flera tjänster som sammanfattar olika nyhetsmediers information för olika

[40] Wikipedia Really Simple Syndication, http://sv.wikipedia.org/wiki/RSS, 2006-09-20

syften och det har inte förekommit några rättsfall kring detta. I allmänhet kan man nog anta att nyhetstjänsterna på nätet ser detta som en möjlighet till gratisreklam och därför inte upplever det som något hot mot deras verksamhet.

4.3.4 Diskussion

Under arbetet med tjänsten Oddsnews.com har vi upplevt att många av spelbolagens XML-feeder i praktiken varit mer ostrukturerade än vad de behövt vara – vilket har gjort att det tagit ungefär lika lång tid att inhämta informationen från deras webbsidor direkt som från deras XML-feeder. Ofta har vi också märkt att man från spelbolagens sida har varit mer mån om att se till att informationen alltid finns tillgänglig och är korrekt på webbsidorna än i sina XML-feedar. Detta har gjort att den teoretiska fördel man ser med XML-parsers framför parsers som använder sig av *regular expressions* har varit mindre i praktiken än i teorin. Man kan tycka att det optimala skulle vara om spelbolagen arbetade fram någon form av standard för hur de presenterar sin xml-feedar för att på så sätt möjliggöra en enkel automatisk jämförelse mellan deras odds. Samtidigt skulle en sådan utveckling göra spelbolagens information enklare att samla in – vilket skulle öppna upp för fler konkurrenter till Oddsnews Ltd. En sådan utveckling skulle alltså inte taget i ett större sammanhang säkert vara till fördel för Oddsnews Ltd.

4.4 Informationsbearbetning

När informationen inhämtats med hjälp av de olika former av extraktorer som beskrivits ovan och har lagrats i databasen är det dags att göra något av den. Det är just bearbetningen och koppling av information från olika källor som är det centrala i Oddsnews Ltd:s affärsidé och genom att hitta nya sätt att presentera information och relatera viss information till annan kan man hitta konkurrensfördelar gentemot andra webbtjänster med liknande innehåll. Nedan kommer vi att diskutera några av de sätt på vilka informationen bearbetas.

4.4.1 Analys och sammanställning av odds och resultat

Den huvudsakliga informationen som samlas in av Oddsnews.com är odds från spelbolag och resultat på spelade matcher. När informationen väl är insamlad relateras informationen till varandra på olika sätt. Det kanske mest uppenbara är direkta jämförelser av spelbolagens odds för att se vilka som ger de bästa oddsen. En något mer komplicerad analys är uträkningar av hur mycket pengar spelbolagen i genomsnitt ger tillbaka till sina spelare. Detta beräknar vi med hjälp av formel 1, där a,b,c är oddset för 1,X respektive 2. Vad vi får ut är ett decimaltal vilken säger hur stor andel av sina satsade pengar en spelare får tillbaka om den satsar på alla tre alternativen samtidigt. Vanligtvis rör sig detta tal någonstans mellan 0,8 och 0,95 – men i vissa fall kan det även överstiga 1 vilket betyder att man kan få en garanterad vinst, ett så kallat ”Surebet”.

Formel 1: $$återbetalning = \frac{1}{\frac{\frac{1}{a}+\frac{1}{b}+\frac{1}{c}}{3}}$$

Återbetalningen räknas ut för varje spelbolag och match. Utifrån detta räknas sedan även ut den genomsnittliga återbetalningen för varje spelbolag. Utifrån ett stort antal odds får vi alltså till slut en siffra som genomsnittligt säger hur mycket varje spelbolag

betalar tillbaka till spelarna och hur mycket det behåller för sig självt. Utifrån oddsen kan vi även beräkna spelbolagens genomsnittliga approximation för att ett visst lag vinner (formel 2).

Formel 2: $$p = \frac{\sum_{i=1}^{n} \frac{1}{o_n}}{n}$$

4.4.2 Prediktion av matchresultat

Grunden för allt spel handlar om hur mycket information man har för att kunna beräkna sannolikheter om framtida utfall. Ju bättre sannolikheter man har – desto större chans att man kan tjäna pengar på sina vad. Här blir alltså möjligheten att dra slutsatser om framtida matchresultat utifrån historiska data en väldigt viktig egenskap. Eftersom Oddsnews genom sitt informationsinsamlade bygger upp en växande databas med historiska data blir en intressant fråga: kan vi utifrån vår information säga något om framtida matchresultat?

För att kunna använda vår historiska statistik har vi skrivit en regressionsalgoritm för att extrapolera dem på framtida matcher. Prediktionsalgoritmen är intressant att diskutera eftersom det är ett tydligt exempel på vad man kan göra när man har mycket insamlad data från spridda platser. Genom att knyta samman dessa resultat kan vi göra avancerade beräkningar utifrån en relativt stor datamängd och på så sätt förädla fram information som är unik på nätet och förhoppningsvis är av tillräckligt hög kvalitet för att locka besökare till webbtjänsten.

I vårt arbete för att utarbeta en algoritm som kan förutsäga framtida matchresultat har vi tagit intryck av examensarbetet *Beräkning av Sannolikheter för Utfall i Fotbollsmatcher – Oddsen på din sida*[41] av Philip Jonsson vid nationalekonomiska institutionen vid Uppsala Universitet. Där pekar han på relevanta faktorer så som lagens aktuella form (ett indexvärde beräknat utifrån dess position i dess x senaste matcher) och vilket lag som spelar på hemmaplan. Den algoritm vi arbetat fram skiljer sig dock radikalt från Jonssons. Huvudsakligen eftersom vår modell försöker skatta ett sannolikt målresultat snarare än att skatta den sannolikaste vinnaren. Det gör också att vår modell är betydligt mer osäker och därför i nuläget snarare ska ses som en funktion för att skapa diskussion och funderingar på webbsidan snarare än ett reellt hjälpmedel för spelare.

Våra beräkningar bygger i grunden på att vi genomför en multipel regressionsmodell på formen $y_i = \beta_1 + \beta_2 x_1 + \beta_3 x_2 + \varepsilon_i$, där y_i är matchresultatet och x är en vektor med relevanta variabler och β är en vektor med regressionskoefficienter. Koefficienterna får vi genom regressionsformeln $b = (x^T x)^{-1}(x^T y)$, där X är en matris med de relevanta variablerna i alla tidigare matcher i samma liga och Y är en kolumnvektor med alla lags matchresultat. De relevanta variabler vi använt är lagens form (en indexvariabel som beräknas med formel 3), motståndarlagets form som vi beräknar på motsvarande sätt, om laget var hemmalag i matchen, lagets genomsnittliga målresultat i ligan fram till

[41] Philip Jonsson, *Beräkning av Sannolikheter för Utfall i Fotbollsmatcher – Oddsen på din sida*, Examensarbete D Nationalekonomiska institutionen, Uppsala Universitet VT 2006

aktuell match och motståndarlagets genomsnittliga målresultat. Slutligen avrundar vi vårt resultat y_i till närmaste heltal.

Formel 3: $f_j = \sum_{i=j-5}^{j-1} r_i \qquad \begin{cases} r = 0 \, vid \, förlust \\ r = 1 \, vid \, oavgjort \\ r = 3 \, vid \, vinst \end{cases}$

För att beräkna allt detta använder vi oss av klassbiblioteket JAMA (A Java Matrix Package)[42] som precis som namnet antyder möjliggör en rad matrisoperationer direkt i java. Detta paket används sedan i vår egna prediktionsklass som beräknar prediktioner utifrån ovan nämnda algoritm och sparar ner dem i databasen tillgängligt för webbapplikationen.

[42] JAMA (A Java Matrix Package), http://math.nist.gov/javanumerics/jama/, 2006-09-20

4.5 Webblagret

Även om den komplexa applikation som ligger bakom webbtjänsten Oddsnews.com är intressant rent tekniskt så är otvivelaktigt den mest centrala delen för webbtjänsten förmågan att presentera data från den bakomliggande applikationen på ett överskådligt sätt. Rent tekniskt består webblagret av en uppsättning jsp- (java server pages), css- (cascading style sheets) och js- (JavaScript) filer. Med hjälp av jsp-filerna skapar webbservern html-filer av den data som applikationens viewlager gör tillgänglig. Dessa html-sidor är sedan det som besökarnas webbläsare renderar på ett sätt som är lättillgängligt för en mänsklig läsare.

Det centrala för webblagret är de jsp-sidor som skapar html-dokumenten. Utan dessa skulle inte webblagret fungera. Den strukturella lösning som har valts för dessa filer är symtomatisk för resten av utvecklingsprojektet. Man har valt att dela upp jsp-informationen på flera filer vilka inkluderas vid behov utifrån en main-fil (main.jsp) som körs i de flesta fall då en sida begärs från servern[43]. Trots den relativt simpla lösningen med flera olika jsp-filer ger detta bättre överblick över koden utan att kräva en högre grad av komplexitet i webblagret – vilket under utvecklingen uppfattades som positivt.

Förenklat fungerar main-filen så att den använder sig av den information som front-controller skapar utifrån de inkommande argumenten och den bakomliggande applikationen (se stycket om Spring MVC för en noggrannare genomgång av denna process). Utifrån vilken information som är tillgänglig avgör sedan main-jsp vilken html-information som ska skickas tillbaka till klienten som skickade förfrågan. Fördelen med detta sätt att skriva webblagret är att vi kan abstrahera bort den bakomliggande applikationen när vi skriver webblagret.

För att göra presentationen så enkel som möjligt utifrån besökarens synvinkel och för att sidan ska bli mer lättunderhållen använder sig webblagret även av css-filer som beskriver hur html-informationen ska struktureras layoutmässigt, samt js-filer vilket gör innehållet dynamiskt – det vill säga möjliggör att delar av sidan kan förändras asynkront utan att all html behöver uppdateras med ny information från servern. Båda dessa tekniker är intressanta och för med sig både för- och nackdelar och dessa diskuteras därför mer ingående i egna avsnitt nedan. I figur 3 kan man se slutresultatet av webblagrets arbete.

[43] I vissa fall har man dock valt alternativa lösningar för att förenkla utvecklingen av avvikande sidor, som till exempel den del av tjänsten som har travodds som definieras i en egen fil.

Figur 3 - Webbsidan Oddsnews.com (2006-10-04)

4.5.1 Asynkron laddning av webblagret (AJAX)

Den ovan beskrivna processen med en jsp-fil som levererar data från front-controllern förutsätter att all html laddas om så fort man vill ändra utseendet på den html-sida besökaren tittar på. Detta har varit det centrala paradigm efter vilket webben har fungerat ända sedan dess tillkomst. En stort brist i detta sätt att uppdatera sidor är dock att det i praktiken ofta bara är vissa delar av en webbsida som ändras när man följer en intern länk. Att ladda om hela sidan tvingar alltså besökaren att ladda om onödigt mycket data och gör att webbsidan ger ett långsammare intryck.

Från 2005 och framåt har det dock utvecklats en teknik för att kunna uppdatera webbsidor asynkront – vilken ofta går under beteckningen AJAX (*Asynchronous JavaScript and XML*). Som namnet antyder använder den JavaScript för att tolka information definierad i XML-filer (*Extensible Markup Language*) för att på klientsidan uppdatera specifika delar av ett html-dokument. Ofta får dock Ajax en bredare betydelse – vilket tydligt märks i följande citat:

> No longer are you forced to wait five seconds a web page to reload every time you click on something. Ajax applications change in real-time. They let you drag boxes around

> instead of clicking on arrows and typing in numbers. They keep page content fresh instead of forcing you to keep hitting Refresh. They show meaningful animations instead of verbose messages.[44]

I denna betydelse blir Ajax snarare ett samlingsbegrepp för de tekniker som (oftast med hjälp av JavaScript) förändrar utseendet av en webbsida utan att ladda om hela sidan. Skillnaden mellan dessa två betydelser blir att den första smalare betydelsen är att Ajax involverar ett anrop till webbservern vilket resulterar i att klienten får ny information i form av en XML-fil och att den andra snarare kan innebära att information som redan finns på klientsidan med hjälp av JavaScript presenteras på annorlunda sätt (exempelvis att tidigare dold information visas samtidigt som annan information döljs). För användaren blir dock denna distinktion mindre intressant och ibland knappast märkbar – varför man oftast använder beteckningen Ajax för båda företeelserna på ett relativt oproblematiskt sätt.

Trots att Ajax har funnits en till och med i Internetsammanhang ganska kort tid – har dock en rad problem med tekniken uppmärksammats från en rad olika håll. Den mest uppenbara invändningen är att Ajax-teknik är beroende av JavaScript och därför inte kommer att fungera i webbläsare som inte kan tolka JavaScript. I praktiken förstår dock alla de mest spridda webbläsare JavaScript varför detta inte är ett stort praktiskt problem. En annan vanlig invändning är användarvänligheten hos lösningar som baserar sig på AJAX. Asynkront laddade sidor leder till tillgänglighetsproblem för funktionshindrade, så som exempelvis blinda vilkas webbläsare ofta inte kan hantera javascript. Även personer med nedsatt syn kan få problem att se vad som händer om endast en liten del av webbsidan har uppdaterats.[45] Utöver att tekniken kan medföra problem för personer med särskilda behov kan Ajax-teknik även försämra indexering av sidor av sökmotorer som exempelvis Google – något som kan försvåra marknadsföringen av en webbtjänst.

Dessa problem med Ajax är något man är tvungen att ta hänsyn till redan under utvecklingen och det är också något som gör att man är tvungen att ta ställning till om man vill använda Ajax i sin applikation eller inte. Under utvecklingen av Oddsnews har man dock ansett att fördelarna överväger nackdelarna och att nackdelarna kan minimeras under utvecklingens gång – varför vissa delar av webbtjänsten använder sig av Ajax-teknik för att göra sidan snabbare och mer lättöverskådlig. Något som är viktigt att poängtera är att man är tvungen att göra en avvägning i varje enskilt fall om asynkron laddning är att föredra framför ett mer traditionellt tillvägagångssätt.

4.5.2 Cascading Style Sheets (css)[46]

Precis som vi tidigare nämnt presenteras precis som alla andra webbsidor Oddsnews.com för besökarna i form av en renderad HTML-fil. HTML står för *Hyper Text Markup Language* och användes från början bara för att definiera textsidor med länkar till andra webbsidor. Allt eftersom webben växte fick HTML-filerna möjlighet att definiera mer innehåll är bara text – exempelvis bilder. Detta har gjort att filerna blev allt mer komplexa och att man velat koda mer avancerad layout för sina webbsidor. Till en början gjordes detta med så kallade Frames – där man definierade hur flera olika

[44] Mahemoff , M., *Ajax Design Patterns*, O'Reilly 2006
[45] Se exempelvis http://www.webaim.org/techniques/ajax/, 2006-10-04
[46] Håkon & Wium Lie, Bert Bos, *Cascading Style Sheets: designing for the Web,*, Addison Wesley, 1999

html-sidor låg i förhållande till varandra i ett webbläsarfönster. Senare började man istället använda tabeller – en ad hoc-lösning eftersom tabeller från början var en definition av ett visst typ av innehåll snarare än en metod skapad för att kunna bygga en avancerad layout. Denna ad hoc-lösning innebar också att själva kodstrukturen i HTML-dokumenten blev allt svårare att överblicka.

Cascading Style Sheets (css) är ett försök att skapa ett alternativ till tabeller. Tanken är att möjliggöra en layout för webbsidor som ligger utanför själva HTML-dokumentet. På så sätt kan man ändra utseendet på en webbsida utan att ändra i själva HTML-koden eller enkelt flytta över ett utseende från en existerande webbsida till en ny.

I praktiken fungerar css så att man definierar definierar id- och/eller klassparametrar åt de html-element för vilka man vill anpassa utseendet. Det man vill anpassa kan vara allt från utseende på text, bakgrundbilder till hur olika delar av sidan ska ligga i förhållande till varandra. Vidare definierar man utseendet för dessa id:n och klasser i antingen en extern .css-fil eller internt i html-dokumentet.

Fördelen med css är att man kan skriva mer semantisk kod (en kod där man använder list-element för listor, tabeller för tabeller osv) samtidigt som man kan presentera koden på ett flexibelt sätt grafiskt. Görs detta på rätt sätt förenklar det även visningen av HTML-dokumentet i exempelvis textbaserade läsare – vilket kan underlätta för blinda och andra i behov av en icke-grafisk presentation. Vi får också en mer uppdelad kodstruktur – där du kan göra stora layoutändringar genom att bara behöva ändra några rader kod.

Css är dock fortfarande en standard i utveckling och ofta tolkar olika webbläsare filerna på olika sätt. Detta gör att en rad speciallösningar är nödvändiga för att layouten verkligen ska bli som man vill. Ibland omöjliggör detta en helt semantisk kod utan kräver att man ändå använder exempelvis tabeller i vissa delar av layouten. Under utvecklingen av Oddsnews har vi därför försökt använda css på så många platser som möjligt, där detta har förenklat utvecklingen och lett till en bättre html-kod, samtidigt som vi undvikit en helt ren implementering då detta snarare skulle gett oss fler än färre problem under utvecklingen.

5. Tekniska lösningar i praktiken

Agile Development, eller Agile Model Driven Development (AMDD)[47] är ett av de senaste årens hetaste ämnen inom programvarusektorn. Syftet med AMDD är att effektivisera processen för utvecklingen av programvara, detta genom att specificera ett visst antal steg som varje projekt går igenom. Eftersom det var intressant för oss att sätta oss in i tänkandet bakom AMDD och Extreme Programming (XP) kommer vi på de följande sidorna att med ett exempel (designen och implementationen av en travfunktion hos Oddsnews) visa hur en konkret utvecklingsprocess inspirerad av Agile-tänkandet kan gå till.

I det första steget skapas en enkel modell av det som ska implementerats. Skillnaden mellan AMDD och andra liknande metoder är att syftet med modellen är att den ska vara "just barely good enough", alltså tillräckligt bra för att fungera till det avsedda syftet, men inte bättre. Detta gör enligt AMDD-förespråkarna att arbetet blir effektivare eftersom man undviker skapandet av stora, komplicerade modeller i förplaneringsstadiet som ändå blir föråldrad allt eftersom projektet fortlöper.

När den grundläggande designen av Oddsnews gjordes av handledarna förutsågs behovet av en funktion som jämför travodds i framtiden, dock prioriterade man att jämföra odds på vanliga 1-x-2-matcher, vilket gjorde att funktionen flyttades till framtiden. När sedan beta-versionen av sajten lanserade i början av augusti 2006 insåg vi snart att ett viktigt önskemål från användarna var just jämförelser av travodds.

5.1 Domänmodellen

En match i den ursprungliga domänmodellen modellerades genom en Event-klass som innehåller information som är gemensam för hela matchen, t.ex. datum, deltagande lag, samt en lista med instanser av klassen BookmakerOdds som innehåller en referens till ett spelbolag samt ett odds (flyttal) för 1, x, respektive 2.

Ett travlopp innehåller ett godtyckligt antal deltagare som var och en har ett eller flera vinnarodds bundet till sig. Modellen för ett sådant blir alltså något annorlunda. Därför flyttades de egenskaper som är gemensamma för alla sorters Event (datum, Kategori, TV-kanal m.fl) till en klass kallad AbstractEvent. AbstractEvent är som namnet antyder just abstrakt, d.v.s. det går inte att skapa instanser av klassen direkt. Sedan gjordes Event-klassen om så att den ärver AbstractEvent, detta för att undvika kodduplicering. En ny klass för att representera en tävling med godtyckligt antal tävlande, var och en med ett eller flera vinnarodds, lades till, kallad MultipleOptionEvent som ärver AbstractEvent. MultipleOptionEvent innehåller en lista med Competitor, som i sin tur innehåller en lista med SingleOdds. SingleOdds är en liten klass som enbart består av ett flyttal som symboliserar oddset och en Bookmaker-enum som symboliserar vilken bookmaker som har just det oddset.

Sedan lades ännu en klass till för att modellera ett travlopp mer specifikt, TrottingEvent, som ärver MultipleOptionEvent.

---

[47] Ambler, S. *Agile Model Driven Development (AMDD*) , http://www.agilemodeling.com/essays/amdd.htm, 2006-09-06

På det här sättet uppnåddes en flexibel modell som enkelt kan byggas ut i framtiden, utan att några onödiga variabler som ännu inte används behövdes läggas till i klasserna. Även om detta på det här stadiet används enbart till Trav, går det att använda MultipleOptionEvent-klassen till egentligen vilken tävling som helst som har ett godtyckligt antal deltagare med en vinnare.

5.2 Enhetstester

När domänmodellen specificerats var nästa steg att få in informationen i databasen. En av hörnstenarna i XP-tänkandet är att koden som skrivs lätt ska kunna testas. Det rekommenderas även starkt att testerna skrivs först, innan någon produktionskod skrivs. Därför skrevs enhets-tester för de olika oddsextraktorerna innan extraktorerna själva skrivits. Som testpaket användes JUnit version 4.1[49]. JUnit är det mest kända testramverket för Java-applikationer och därför var valet av detta självklart. Nytt i version 4 av JUnit är att man kan använda s.k. Annotations, en ny funktion för Java 1.5, för att definiera sina testfall. Detta utnyttjades när testerna skrev för att få en mer strukturerad och överblickbar kod.

Testfallet för Unibet-extraktorn såg till exempel ut så här:

```
@Before
public void loadHtmlFile() {
     testHtml=getContents(new
File("./src/com/oddsnews/trotting/test/unibetTest.jsp"));
     testHtml2=getContents(new
File("./src/com/oddsnews/trotting/test/unibetTest2.jsp"));
}
```

@Before-annoteringen definierar att metoden ska köras innan testerna körs. Här laddas en lokalt lagrad websida innehållande ett eller flera travlopp in och sparas i två strängvariabler för att senare användas i testerna.

```
@Before
public void setupTestContestants() {
     race=new TrottingEvent();
     race.setRaceNumber(4);
     race.setLocation(TrottingLocation.JAGERSRO);
     try {
           date = dateFormat.parse("060903 14:00");
     } catch (ParseException e) {}
     race.setDate(date);

     race.setCategory(Category.TRAV);

     Competitor comp=new Competitor("Thunderbolt");
     comp.addOdds(new SingleOdds(Bookmaker.UNIBET, new Float(6.50)));
     race.addCompetitor(comp);
     comp=new Competitor("Target Player");
     ...
     ... (Och så vidare)
```

[49] Informationen i detta avsnitt kommer om inte annat anges från sidan: ”JUnit, Testing Resources for Extreme Programming”, http://www.junit.org/index.htm, 2006-09-01

```
}
```

Eftersom extraktorernas syfte är att utvinna informationen som finns tillgänglig på websidan och spara ner den i databasen, måste programmeraren när denne skriver testet först definiera vad som är "rätt" output från testet. Detta görs i en annan metod annoterad med @Before. Här skapas den objektstruktur som den fungerande extraktorn förväntas skapa utifrån testsidan. Ett objekt TrottingEvent skapas och en dess egenskaper sätts till önskade värden. Sedan läggs de tävlande till en efter en, med rätt odds och spelbolag definierade. Ännu ett testlopp definierades sedan för att testa hur extraktorn hanterar lopp med en eller flera strykningar (hästar som inte kommer till start).

Efter att det rätta svaren har kodats kan själva testet skrivas. I testet körs extraktorn på det sätt som den kommer att användas i programmet, sedan jämförs de fördefinierade svaren mot de som kommer från extraktorn. Är svaren lika så klarar extraktorn de krav som programmeraren har ställt upp, och därigenom är jobbet klart. Fördelen med enhetstester skrivna på det här sättet är att det är mycket lätt att köra testet efter varje refactoring av klassen, man behöver alltså aldrig vara orolig för att en ändring i koden förstör det som testas, utan klarar klassen testet så fungerar den som är tänkt. Detta är självklart givet att man skrivit ett så fullständigt test som möjligt, så att det verkligen testar de faktorer som är viktiga. Är testet fel eller ofullständigt kan klassen också ge ett missvisande resultat.

```
@Test
public void parseHtml() {
     unibet.setTestString(testHtml);
     List<TrottingEvent> raceList=unibet.parseOdds();
     assertNotNull("No competitors from extractor!", raceList);
     List<Competitor> competitors=raceList.get(0).getCompetitors();
     List<Competitor> expected=race.getCompetitors();

     for (int i=0; i < competitors.size();i++) {
           assertEquals("Competitors not the same.." ,
expected.get(i).getName(),competitors.get(i).getName());
     }

     assertEquals("Category not the same",
race.getCategory(),raceList.get(0).getCategory());
     assertEquals("Date incorrect", race.getDate(),
raceList.get(0).getDate());
     assertEquals("Location incorrect",
race.getLocation(),raceList.get(0).getLocation());
     assertEquals("Race number incorrect",
race.getLocation(),raceList.get(0).getLocation());
     log.info(raceList.get(0).toString());
     log.info("Found " + raceList.size() + " races");

}
```

Ovan ses en typisk Junit-testmetod. Metoden är annoterad med @Test för att symbolisera att det är en metod som testar någon funktion hos klassen. JUnit definierar en mängd metoder som börjar med assert, t.ex. assertEquals(), som används för att jämföra det förväntade objektet med det som kommer ut ur extraktorn. I metoden ovan

jämförs de faktorer som ansågs viktiga och ett felmeddelande skrivs ut om de inte är lika. Liknande testfall skrevs för de två andra spelbolag som valdes att vara med i oddsjämförelsen, Expekt och ATG.

5.3 Extraktorerna

Extraktorer skrev alltså för tre företag, Unibet, Expekt och ATG. För Expekt fanns en xml-feed att tillgå med oddsen definierade enligt en klart definierad mall, detta gjorde den extraktorn lättare att skriva. Unibet och ATG hade inga motsvarande XML-feeds, därför blev det nödvändigt att läsa av deras websidor direkt och plocka ut den intressanta informationen därifrån. Detta gjordes med hjälp av reguljära uttryck, som gås igenom mer ingående i en annan del av uppsatsen. Utmaningen med att använda reguljära uttryck ligger i att hitta mönster i HTML-koden som gör att man kan programmera datorn att plocka ut t.ex. ett odds från rätt plats varje gång, oavsett vilken siffra som faktiskt står där. I xml-filer är detta lätt eftersom de har en syntax som bara beskriver egenskaper hos de objekt som finns med, det säger ingenting om hur denna information ska presenteras. I html-filer så blandas däremot information om objekten med layoutinformation, vilket gör extraktionen svårare, eftersom datorn inte ”ser” layoten på samma sätt som vi människor gör utan hanterar HTML-koden direkt. Därigenom kan en extraktor baserad på reguljära uttryck skrivas för vilken sida som helst, givet att man vet vilken information som är intressant och ska plockas ut.

Alla extraktorer implementerar ett interface kallat TrottingSource, som enbart definierar en metod, List<TrottingEvent> parseOdds(). Genom att definiera interface på detta sätt är det möjligt att använda Springs dependency injection för att "trycka" in extraktorerna i den klass som har ansvaret att samla ihop informationen från de olika spelbolagen.

Det största problemet som uppstod när oddsen från de olika bolagen skulle sammanställas var att fundera ut hur datorn skulle kunna avgöra när oddsen kommer från samma häst. För den vanliga oddsparsningen definierades en enum-klass med Mappningar för alla tillgängliga lag, detta för att kunna ta hand om alternativa stavningar av lagnamn. För travet kändes denna metod obrukbar, eftersom det vore ett enormt och tidsödande jobb att mata in alla de hästar som tävlar i de olika travloppen. Istället beslutades att utgå från startnumret när hästarna identifierades, detta är något som är gemensamt för alla spelbolag, även om de stavar namnen på de olika ekipagen olika.

Det andra problemet för sammanställningen av oddsen var hur vi automatiskt skulle kunna avgöra vilka lopp som skulle definieras som samma, och därmed läggas in för att jämföras. Detta gjordes med hjälp av en enum, TrottingLocation, som definierar vilka tillåtna alternativ för travbanor som finns. Sedan jämför sammanställningsklassen Location, datum och loppnummer, om alla dessa tre stämmer överens så läggs de i samma lopp i databasen.

```
private void addOrUpdate(TrottingSource t) {
     List <TrottingEvent> eventList=t.parseOdds();
     log.info("Found: "+ eventList.size() + " races");
     TrottingEvent existing=null;
     try {
           for (TrottingEvent event : eventList) {
                existing=trottingDAO.findUniqueRace(event.getDate(),
event.getRaceNumber(), event.getLocation());
                if (existing==null) { // Race does not exist
                     log.debug("Adding race..");
                     trottingDAO.makePersistent(event);
                }
                else {
                     log.debug("Merging races");

     trottingDAO.makePersistent(updateOdds(existing,event));
                }
           }
     } catch (Throwable error) {
           log.warn(error.toString());
     }
}
```

För travet är det i högsta grad intressant att se hur ett odds har förändrats över tid. Därför lades en mycket enkel "historik"-funktion in i SingleOdds-klassen. Vid varje uppdatering läggs oddset från den föregående uppdateringen i en variabel kallad oldOdds. När websidan sedan visas finns metoder kallad isUp() och isDown() definierade för SingleOdds, som jämför det aktuella med det föregående oddset.

Exempel på metod som jämför aktuellt med föregående odds:

```
public boolean isUp() {
     return (oldOdds < odds);
}
```

Denna funktion skulle självklart kunna göras mycket mer avancerad, men för att följa AMDD:s "så enkel som möjligt men inte enklare"-filosofi implementerades funktionen på detta sätt. Skulle användarna sedan visa önskemål för en mer avancerad historikfunktion kan den implementeras då.

5.3.1 Nya funktioner i Java 1.5

Oddsnews använder flitigt många av de nya funktionerna i Java 1.5, en av dessa är introduktionen av typsäkra enums. Enums används för att definiera en statisk samling av objekt, d.v.s. objekt som inte kommer att ändras under körningens gång. Genom att definiera enums kan man göra tillgängligt en lista med tillåtna värden för en datatyp. Tidigare har många programmerare använt int-variabler med flaggorna static final, d.v.s. att de inte kan ändras och att de är gemensamma för alla instanser av klassen. Den största nackdelen med det är att de just inte är typsäkra, vilket kan illustreras med följande exempel:

```
class Places {
     final static int STOCKHOLM=1;
     final static int GOTEBORG=2;
}

class Person {
     private String name;
     private int city;
}
```

Definieras Places-klassen på detta sätt kan vi skriva city=Places.STOCKHOLM för att symbolisera att en person bor i Stockholm. Men det är också tillåtet att skriva city=873, men vad betyder detta då? Detta exempel illustrerar på ett tydligt sätt problemet med icke-typsäkra enumerationer.

Det andra sättet som problemet löstes på tidigare var genom att använda någon variant av "Type-safe enum pattern"[51] , som var ett vanligt design pattern innan Java 1.5.

Skulle exemplet ovan istället göras om med enums i Java 1.5, kunde det se ut så här:

```
enum Places {
     STOCKHOLM, GOTEBORG;
}
class Person {
     private String name;
     private Places city;
}
```

Nu kan vi fortfarande skriva city=Places.STOCKHOLM, men vi kan inte tilldela city-variablen något annat värde än de som är definierade i enumen.

Detta utnyttjas för travet genom att en enum kallad TrottingLocation definieras, som innehåller de travbanor för vilka lopp ska hämtas in.

---

[50] http://java.sun.com/features/2003/05/bloch_qa.html
[51] http://java.sun.com/developer/Books/shiftintojava/page1.html

```
public enum TrottingLocation {
JAGERSRO("Jägersro","J"),
...
OREBRO("Örebro","Ö");

private String name;
private String banKod;

private TrottingLocation(String name, String banKod) {
     this.name = name;
     this.banKod= banKod;
}
```

När extraktorerna läser in ett lopp från något av bolagens hemsida, använder de en metod Find(String name) i TrottingLocation-enumen, för att se om det loppet körs på någon av de banor som vi är intresserade av, t.ex. så utelämnas de minsta banorna ur TrottingLocation, eftersom intresset av dessa bland användarna inte är särskilt stort. Är den det, returneras en TrottingLocation-enum och resten av informationen om loppet extraheras.

En annan viktig ny funktion som användes flitigt är Generics. Generics ger programmeraren en möjlighet att berätta för kompilatorn vilken datatyp en Collection, d.v.s. en lista, en Map eller liknande, innehåller. Ett exempel från Java-dokumentationen visar hur detta bidrar till en mer överskådlig och säkrare kod:

Innan Java 1.5[52]

```
List myIntList = new LinkedList(); // 1
myIntList.add(new Integer(0)); // 2
Integer x = (Integer) myIntList.iterator().next(); // 3
```

Med Generics:

```
List<Integer> myIntList = new LinkedList<Integer>(); // 1'
myIntList.add(new Integer(0)); //2'
Integer x = myIntList.iterator().next(); // 3'
```

I det andra exemplet ser vi hur programmeraren, genom att denne på första raden specificerar att listan bara får innehålla objekt av typen ”Integer”, introducerar mer information om listan för kompilatorn. Detta medför att kompilatorn vid kompileringen kan kontrollera att programmeraren i resten av koden enbart försöker plocka ut Integers ur listan. I det första exemplet däremot kan listan lagra alla möjliga sorters Objekt, vilket gör att castningen på rad 3 från Object till Integer kan kasta ett RuntimeException under körningen om objektet i listan skulle visa sig vara något annat.

[52] Gilad Bracha, ”Generics in the Java programming Language”, http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf

5.4 Schemaläggning av oddsinhämtandet

För de vanliga matchoddsen bestämde vi oss i början av utvecklingen att det var tillräckligt att läsa av dessa varannan timme, detta eftersom de inte förändras särskilt ofta. Inhämtandet av travoddsen ställde andra krav eftersom dessa odds ändras väldigt ofta, av ett par anledningar. Dels så finns något som kallas "Tattersalls regel 4" som säger att om en häst stryks av någon anledning så kommer oddsen på alla andra hästar att reduceras med en förutbestämd summa. En annan anledning är att ATG är en oddsbörs, d.v.s. användarna satsar mot varandra, vilket gör att oddsen i teorin förändras varje gång någon spelar på loppet.

Dessa faktorer gjorde att det fanns ett krav på att oddsen uppdaterades så ofta som möjligt. Dock skulle uppdateringar som gjordes för ofta belasta servern och nätverket onödigt mycket, det krävdes alltså att vi hittade en kompromiss.

Problemet löstes genom att sätta olika uppdateringsfrekvenser beroende på hur lång tid det är kvar till nästa lopp. Även här användes Enums för att definiera vilka uppdateringsfrekvenser som finns och hur långa dessa ska vara. Efter studier bestämdes att tre nivåer var tillräckligt, dessa kallades då NORMAL, FASTER respektive FASTEST. Varje av dessa nivåer innehåller en int-variabel som representerar hur många sekunder systemet ska vänta mellan varje uppdatering om man kör på just den uppdateringsperioden. Normal uppdateringsfrekvens sattes till 2700 s (45 min), den snabbare till 1200 s (20 min) och den snabbaste till 300 s (5 min).

Genom att läsa ut vilka lopp som körs idag ur databasen kunde det sedan definieras regler för hur ofta uppdateringen ska ske. Finns inget aktuellt lopp idag väljer systemet den långsammaste uppdateringsperioden, eftersom oddsen då antagligen inte kommer att förändras lika mycket. Kommer ett eller flera lopp att köras idag, men inte inom den närmaste timmen, så snabbar systemet istället upp till FASTER-nivån. Är tiden till nästa lopp till slut mindre än en timme så snabbas uppdateringen på ytterligare till FASTEST-nivån.

Klassen som har ansvar för oddsuppdateringarna schemalades sedan med hjälp av Springs ScheduledExecutor-API att exekveras ungefär var 5:e minut. I metoden run() jämförs sedan den aktuella tiden just nu med den tiden till vilken nästa uppdateringen är schemalagd. Om den aktuella tiden är efter nästa schemalagda uppdatering så körs hela processen, annars gör metoden ingenting mer.

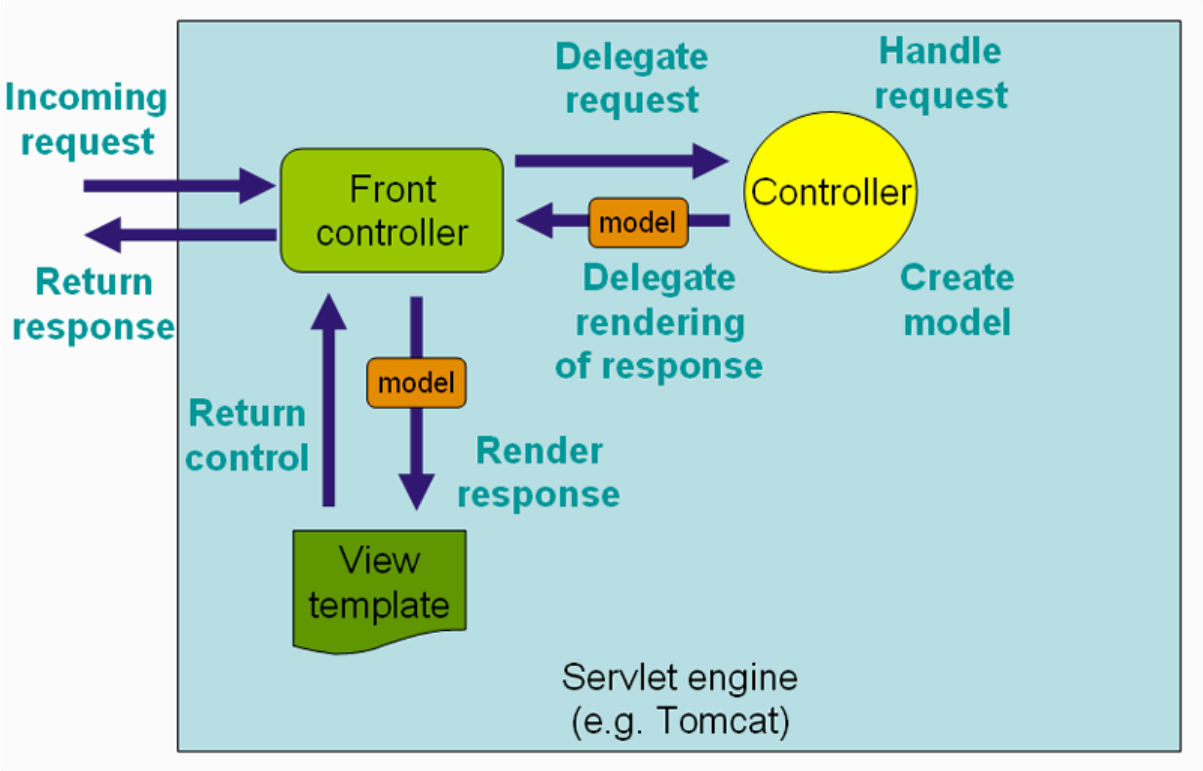
```
public void run(){
     Date now=Calendar.getInstance().getTime();
     Date nextUpdate=trottingDAO.getNextUpdate();

     if (now.after(nextUpdate)) {
           updatePeriod();
           for (TrottingSource t : trottingSources) {
                log.info("Updating trotting odds from " +
t.getBookmaker().toString());
                addOrUpdate(t);
           }
           updateNumberOfEvents();
     }
}
```

Genom att på detta sätt använda enums och en metod som, utifrån innehållet i databasen, returnerar en lämplig takt att uppdatera oddsen i, uppnåddes en kompromiss mellan snabbhet i uppdateringarna och resursanvändning på servern.

5.5 Att visa informationen på webbsidan

Nu när informationen har samlats in och strukturerats på ett lämpligt sätt samt lagrats i databasen var det dags att skriva den kod som behövs för att visa oddsen på sidan. Hela Oddsnews använder Springs MVC-del för att visa informationen.

Grunden i Spring MVC ligger i användningen av en dispatcher-servlet, som har som uppgift att hantera requests från klienter och vidarebefordra dessa till lämplig kontroller. Hela arbetsprocessen kan på en hög nivå schematiskt beskrivas som i figur 4 nedan:



Figur 4 - Arbetsflödet i Spring MVC [53]

Dispatcherservleten tar alltså emot anropet från klienten och skickar den vidare till lämplig controller. Controllern får sedan i uppgift att, givet av vad som anges i anropet, plocka fram den information som behövs, t.ex. ur databasen. Den returnerar sedan ett antal objekt med svaret på klientens anrop till dispatchern. Dispatchern använder sedan den definierade vyn (jsp:er i Oddsnews fall) och renderar ett svar i html-format, vilket sedan skickas till användaren.

---

[53] Johnson, R. Et.al (2006)., *"The Spring framework – Reference Documentation"*, *Kapitel* 13, http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html, 2006-10-16

Spring MVC är som namnet antyder en implementering av ett design pattern som kallas ”Model – View – Controller”, eller mer specifikt ”Front controller”. Med hjälp av detta separeras de olika ansvarsområdena i applikationen effektivt, vilket gör att man inte behöver bry sig om hur sidan ska se ut i controller-koden, hur datan ska plockas ur databasen i jsp-koden och så vidare. Just MVC-mönstret har Spring MVC gemensamt med många andra framework, andra exempel är t.ex. Struts och Webwork.

Spring har ett stort antal färdiga controllers som kan användas vid olika behov. Eftersom nästan alla sidor på Oddsnews innehåller en sökruta för att söka efter nyheter, matcher och liknande, så behöver de controllers som används kunna hantera HTML-formulär. För travsidan valdes att använda klassen SimpleFormController. Trots namnet kan denna användas till mycket avancerade formulär. SimpleFormController definierar en enda abstrakt metod som programmeraren måste implementera:

```
public ModelAndView onSubmit(final HttpServletRequest request,
HttpServletResponse response, Object command, BindException errors)
throws Exception {
```

Denna metod körs varje gång en klient ”skickar in” formuläret som standard. Men eftersom vi även vill tillåta att användaren inte söker efter något, användes en funktion i Spring som gör att onSubmit körs varje gång den aktuella sidan efterfrågas:

```
protected boolean isFormSubmission(HttpServletRequest request) {
     return true;
}
```

I onSubmit analyseras sedan informationen i request-headern från användaren, utefter detta avgörs sedan vilken information som ska läggas i ModelAndView objektet.

5.5.1 Tillämpning av AJAX på Oddsnews.com

På Oddsnews.com använder vi både den typ av AJAX som verkligen använder sig av asynkron laddning samt sådan som snarare strukturerar redan hämtad information på ett bättre sätt. Nedan beskrivs dock bara det fall där vi gör asynkrona anrop – eftersom detta är intressantast ur ett datavetenskapligt perspektiv.

Den tillämpning av AJAX som använts på webbtjänsten Oddsnews bygger på biblioteket DWR (Direct Web Remoting). DWR förenklar ajaximplementationer i java-applikationer och består av en javascriptbaserad klientdel och en serverdel skriven i java. I JavaScript-filen begärs data och data som returnerats monteras i den laddade HTML-Dom:en. Exempelvis kan det se ut så här:

```
1 function postEventComment(type) {
2    […]
3    newsComment.postComment(url,postedBy,"",message,type,
4          {callback:
5          function(str) {
6               if (str!=null) {
7                    /* Show status message in div */
8                      DWRUtil.setValue("ev_postResult", str);
9                    initEventComments(url,headline); //Update posts
10                   […]
```

```
11              }
12          },
13          timeout:10000,
14          errorHandler:function() {
15              DWRUtil.setValue("ev_postResult", "Kunde inte
kommunicera med servern");
16          }
17    });
18    }
19 }
```

Rad 3 kallar med hjälp av JavaScript funktionen postComment hos javaobjektet NewsComment. Detta möjliggörs av DWR och en mellanliggande XML-fil där det definieras vilka objekt som är åtkomliga och vilka av dessas metoder som kan användas asynkront. För att anropet ovan ska vara möjligt behövs följande i definitionsfilen dwr.xml:

```
<create creator="spring" javascript="newsComment">
     <param name="beanName" value="newsComment"/>
     <include method="postComment"/>
     <include method="getComments"/>
     <include method="getNumberOfPosts"/>
</create>
```

I dwr.xml definierar vi alltså bönan som kallas när vi skriver `newsComment.postComment([…])` – vilket i detta fall är bönan:

```
<bean id="newsComment"
     class="com.oddsnews.comment.CommentHandler">
     <property name="commentedItemDAO" ref="commentedItemDAO" />
     <property name="mainNews" ref="mainNews" />
     <property name="successString" value="Postade meddelandet" />
     <property name="failureString"
          value="Kunde inte posta meddelandet" />
</bean>
```

Denna böna definieras i filen ApplicationContext.xml och det är alltså här vi definierar vilken javaklass som ska köras då vi kallar `newsComment.postComment([…])` i JavaScriptet. I objektet Commenthandler har vi slutligen metoden som körs, i vilken vi lägger in kommentaren i databasen och returnerar ett svar som dwr skickar tillbaka till klienten som gjort det asynkrona anropet.

Svaret fångas på klientsidan upp av en JavaScript-funtion, en så kallad callback-funktion som definieras i rad 5-12 i javascriptet. Denna funktion är alltså både ansvarig för att fånga upp svaret och för att montera informationen på rätt plats i vår HTML-DOM. Utöver callback-funktionen definierar vi även hur länge klienten ska vänta på svar från servern (timeout, rad 13) samt en funktion som körs om något går fel (rad 14-16).

```
ModelAndView model = new ModelAndView();
```

```
if (request.getParameter("category")!=null &&
!request.getParameter("category").equals("")) {
     int categoryId =
Integer.parseInt(request.getParameter("category"))
     model.addObject("races",trottingDAO.findByCategory(Category.getCat
egoryById(categoryId)));
}
```

I koden ovan undersöker vi om requestheadern innehåller en varibel som heter Category, i så fall hämtas de travlopp som finns i just den kategorin (t.ex. V75,V65) ut ur databasen och läggs in i Modellen.

Just MVC-delen är ett av de få delar av Spring där koden direkt beror på Springs API:n. Det kontrollern gör är konceptuellt väldigt enkelt, den plockar upp antingen en lista med tillgängliga lopp ur databasen, alternativt en enskilt lopp för att visa det för besökaren. Den passar även på att göra en del andra saker, t.ex. att söka fram lämpliga nyheter för kategorin och välja ut lämpliga banners att visa för användaren.

5.6 Administrationsinterface[54]

TrottingControllern är ett exempel på en väldigt enkel form-controller. En mer avancerad lösning krävdes när ett webbaserat administrationsinterface till Oddsnews skulle skrivas. På Oddsnews finns så kallade infosidor för varje spelbolag. Detta är en sida där användarna snabbt kan se information om vilka villkor de olika bolagen erbjuder, och de ges även möjlighet att snabbt jämföra mellan de olika. Ett krav innan designen för dessa sidor påbörjades var att de skulle kunna ändras av personer utan programmerings- och webdesignserfarenhet, att ändra direkt i koden eller skriva egna HTML-sidor var alltså inget alternativ.

Utvecklingsprocessen för admin-sidorna liknade i stora delar den som användes för trav-delen: Först undersöktes vilken information om spelbolagen som var intressant för användarna. Utifrån detta skissades sedan en enkel domänmodell upp på ett papper, varefter denna implementerades i javakod. Ett DAO-objekt skrevs för att kunna spara och hämta ur infon ur databasen.

Sedan behövdes ett sätt för att få den texten som användaren skrev in om ett bolag via admin-sidorna att integreras i BookmakerInfo-objektet. Utan Spring eller ett liknande framework skulle vi här behöva kolla de variabler som finns i POST-förfrågan från användaren och sedan skriva kod för varje variabel som vi ville ”översätta” från fältet på websidan till motsvarande variabel i domänmodellen. Ett utdrag för hur sådan kod skulle kunnat se ut ges nedan:

```
BookmakerInfo info=new BookmakerInfo();
if (request.getParameter("bookmakerName")!=null &&
!request.getParameter("bookmakerName").equals("")) {
```

[54] All information om Spring i detta avsnitt är hämtat från ”The Spring Framework – Reference Manual”, kapitel 13. http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html#mvc-formtaglib, 2006-09-27

```
     info.setBookmaker(Bookmaker.findByName(request.getParameter("bookm
akerName");
}
```

Liknande kod hade sedan behövts skrivas för alla variabler i BookmakerInfo-klassen som vi vill ha möjlighet att sätta från websidan. Eftersom antalet variabler är en bra bit över 20 är det tydligt att det skulle krävas mycket jobb, och inte minst kod, för att göra detta möjligt.

Istället användes Springs form-bibliotek. Detta är en Tag som används för att direkt från en html-sida översätta strängarna i formuläret till respektive variabel i domänobjektet. Spring tar även hand om eventuella typecasts, d.v.s översättningar från strängar till andra datatyper. Alla headers i POST-requesten är ju i praktiken strängar, och vill vi översätta dessa t.ex. till ett flyttal i domänmodellen måste vi själva göra den omvandlingen, t.ex. med Float.parseFloat() eller liknande, givet att vi inte använder formtag-biblioteket.

```
<%@ taglib prefix="spring" uri="http://www.springframework.org/tags"%>
<%@ taglib prefix="form"
uri="http://www.springframework.org/tags/form" %>
.
.
.
<form:form commandName="bookmakerInfo">
     <form:errors path="*" />

     <table border=1 cellPadding=10>
         <tr>
             <td>Företagets Namn</td>
             <td><form:input size="40" path="legalName" /></td>
         </tr>
         <tr>
             <td>Varumärkesnamn</td>
             <td><form:input size="40" path="brandName" /></td>
         </tr>
         <tr>
..
<form:checkbox path="swedishSupport"/>
..
</form>
```

Det sista exemplet, med path=”swedishSupport” visar hur vi kan binda en kryssruta (checkbox) i HTML-formuläret till en boolean-variabel i BookmakerInfo-objektet.

När vi sedan i Controller vill ha tillgång till BookmakerInfo-objektet som har skapats i formuläret använder görs det enkelt på en enda rad i onSubmit-metoden:

```
public ModelAndView onSubmit(HttpServletRequest request,
HttpServletResponse response, Object command, BindException errors)
     throws Exception {
           BookmakerInfo info=(BookmakerInfo)command;
```

Det finns även möjlighet att validera det användaren matar in på olika sätt. Detta görs genom att skriva en klass som implementerar interfacet Validator. Detta ansågs dock i vårt fall inte behövas, eftersom de som använder sidan är kunniga i vad som förväntas i de olika fälten.

Genom att på det här sättet binda formulärfält till ett objekt, och sedan lagra detta i databasen går det att lägga till info från admin-interfacet. Men en annan funktion som behövdes var möjligheten att ändra i ett objekt som redan finns. Även här kan Spring hjälpa till, genom att implementera metoden formBackingObject(`HttpServletRequest request`) kan vi ange vilken information som ska ligga i formuläret från början. Nedan ges ett exempel på hur den metoden ser ut för admin-funktionen.

```
// Fetch the info we have about a bookie from the database
public Object formBackingObject(HttpServletRequest request) throws
ServletException
     {
     if (request.getParameter("bookmaker")!=null) {
          final Bookmaker bookmaker =
Bookmaker.getBookmakerByName(request.getParameter("bookmaker").replace
('+',' '));
          BookmakerInfo info=infoDAO.findByBookmaker(bookmaker,
false); // Search all bookies
          if (info!=null) {
               return info;
          }
          else {
               BookmakerInfo b= new BookmakerInfo(bookmaker);
               // Set some default values
               b.setMaximumBet(new Double(-1));
               b.setMinimumBet(new Double(-1));
               b.setMaximumWinnings(new Double(-1));
               b.setMinimumDepositAmount(new Double(-1));
               b.setVisible(false);
               return b;
          }
     }
     return null;
}
```

5.7 Banner-systemet

Oddsnews affärsmodell bygger på att användarna registrerar sig hos spelbolagen efter att ha blivit hänvisade dit från vår sida. Sedan ger spelbolagen en viss procent av summan de tjänar på ”våra” användare i provision till Oddsnews. Det görs rent praktiskt antingen genom att användaren klickar direkt på oddsen på en sida, eller klickar på en s.k. ”banner”, d.v.s. en reklambild, som finns på varje sida.

Idag finns det några svenska företag som har som affärsidé att tillhandahålla färdiga reklamlösningar för webbplatser, t.ex. Tradedoubler (www.tradedoubler.se) och Double (double.se). Hos dessa anmäler man sig till så kallade reklamprogram för företag som man tror kommer att passa in på sin sida. Sedan är det upp till företaget som äger reklamen att godkänna eller inte godkänna ansökan. Godkänns ansökan får man en bit html-kod som man lägger in på sidan, sedan sköts all visning av reklamen samt statistikinsamlande av Tradedoubler själva. Hos Tradedoubler kan man via ett enkelt

webgränssnitt ställa in vilka annonser som ska roteras, vilka som ska visas mer och så vidare.

Hos Oddsnews hade det, eftersom vi prioriterat effektivitet och användandet av färdiga lösningar under utvecklingsprocessen, självklart varit att föredra om ett liknande system som Tradedoubler funnits för de olika spelbolagen, men verkligheten ser tyvärr annorlunda ut. Varje spelbolag har sitt eget system för att visa och samla in statistik över reklam, vilket gör att Oddsnews tvingas ha konton hos ungefär 15 olika spelbolag. Detta gör administrationen av dessa ineffektiv, så för att göra det lättare bestämdes att det skulle utvecklas ett administrationssystem även för reklam. Följande krav ställdes upp[55]:

- Via ett web-baserat administrationssystem ska Oddsnews administratörer kunna lägga in HTML-kod för banners från de olika spelbolagen, vilka då ska sparas i databasen.

- När en användare går in på sidan ska banners väljas ut slumpmässigt ur databasen och visas på sidan. Systemet ska inte visa två banners från samma bolag på samma sida.

- Administratören ska kunna specificera vilket format (storlek) denne vill ha på reklamen på sidan, systemet ska sedan dra en slumpmässig banner från alla med korrekt storlek. Det ska gå att sätta övre respektive undre gränser på storlek, så att reklam med storlek inom användardefinierade intervall kan tas fram ur databasen.

- Samtidigt ska systemet samla in statistik över reklamen, minst så ska statistik över antalet visningar samt antalet klick samlas in. Genom dessa två mått är det sedan enkelt att beräkna vilka annonser som går bäst på sidan.

- Godtyckligt antal annonser ska kunnas läggas till samt tas bort på ett enkelt sätt.

5.7.1 Implementation

En Banner modellerades med hjälp av ett Banner-objekt, som innehåller en Sträng där HTML-koden klistras in. Den innehåller variabler för antalet visningar, antalet klick samt storlek. En Bookmaker-enum finns i varje Banner-objekt för att visa till vilket spelbolag en viss reklam hör, på det sättet kan vi undvika att flera banners från samma företag visas samtidigt på sidan.

Sedan skrevs en DAO för att kunna plocka ut reklam ur databasen. Interfacet BannerDAO definierar tre metoder:

```
public interface BannerDAO extends GenericDAO<Banner, Long>{
    List<Banner> findByBookmaker(Bookmaker bookmaker);
    List<Banner> findBySize(int x, int y);
    Banner findRandomBySize(int x, int y, int maxDiffX, int maxDiffY);
}
```

[55] Denna typ av funktionella krav kallas också ofta ”Use Cases”, se till exempel http://en.wikipedia.org/wiki/Use_case

findByBookmaker() returnerar alla banners från ett visst spelbolag, detta används i administrationssystemet för att visa banners. findRandomBySize() används sedan i service-lagret för att avgöra vilka banners som ska visas på sidan. Genom att sätta den önskade storleken i x- och y-led samt hur mycket storleken maximalt får avvika från dessa två kan man plocka ut banners även om de inte är precis lika stora. Ett vanligt format för banners är t.ex. 468x60 pixlar, men det finns även en del som är 480x60 pixlar. Genom att sätta x till 470, y till 60, maxDiffX till 20 och maxDiffY till 0 kan man därmed få ut alla banners i det intervallet. Designen på sidan tillåter en del differens i reklamstorlek, och med hjälp av findRandomBySize() går det att anpassa specifikt för varje plats hur mycket som tillåts.

Ett annat krav som ställdes var att systemet skulle föra statistik över hur olika reklam-banners fungerar. Antalet visningar var lätt, eftersom varje banner som plockats ut ur databasen av viewService-lagret kommer att visas på sidan kan den logiken skötas där. Antalet klick är däremot knepigare: eftersom länkarna som reklamen pekar mot inte ligger på vår server finns det inget direkt sätt för oss att spåra antalet klick. En vanlig lösning på problemet är att man skriver en redirect-servlet, alltså en servlet på den egna servern som tar ett argument som request-parameter och sedan skickar användaren vidare till den externa sidan. Problemet med denna lösning i detta fall är att man då blivit tvungen att ändra i html-koden för reklamen, de skulle alltså inte direkt ha kunnat klistrats in i administrationssystemet. Ett annat problem är att spelbolagen kontrollerar den s.k. ”referrer”-headern i html-requesten, vilket gör att de skulle få trafik från sidor som inte finns i deras system. Detta skulle förmodligen göra att de skulle se trafiken som falsk, och därmed inte ge Oddsnews provision för de användare som värvas.

Därför bestämdes att det istället skulle användas en AJAX-lösning för klick-spårningen, fördelen med det är att vi då inte behöver göra några ändringar i bannerkoden. En simpel BannerClick()-klass skrevs på servern som kallas med hjälp av ett JavaScript från klientsidan. Metodens addClick:s enda uppgift är att lägga till ett klick för en viss banner och spara resultatet i databasen.

```
public boolean addClick(Long id) {
     Banner banner=bannerDAO.findById(id,true);
     if (banner!=null) {
           banner.addClick();
           bannerDAO.makePersistent(banner);
           return true;
     }
     return false;
}
```

Koden som kallar på denna metod i html-en ser sedan ut på detta sätt:

```
<div class="flash" id="titleadd"
onClick='banner.addClick(${banners.topBanner.id},function(str) {});'>
     <c:out value='${banners.topBanner.html}' escapeXml='false'/>
</div>
```

När användaren klickar på det som finns inom div-taggen (d.v.s. själva bannern), kallas addClick-metoden på servern med den aktuella Bannerns id som argument. Detta sker helt transparent för användaren, och spelbolagen ser den ursprungliga adressen som referrer i sina loggar, vilket var precis det problemet som skulle lösas.

5.7.2 Möjliga förbättringar

Banner-systemet fungerar bra i dess nuvarande form, men det saknar en del funktioner som skulle kunna vara nyttiga. Till exempel skulle man vilja kunna vikta olika banners beroende på användarens önskemål, någon vecka kanske det bestäms att Oddsnews vill köra en kampanj för ett visst bolag. En annan användbar funktion skulle vara att systemet självt anpassar vilka banners som visas utifrån vilka som går bra, det vill säga genererar många klick. Detta skulle kunna göras med hjälp av statistiska metoder, så kallad Data Mining.

6. Utvärdering av lösningar

I följande avsnitt utvärderar vi de konkret tekniska lösningarna vi valt för webbapplikationen från en rad olika perspektiv – dels utvärderar vi valet av serverprogramvara utifrån ett tekniskt perspektiv, dels gör vi en kort studie av hur väl webblagret fungerat utifrån ett gränssnittsperspektiv samt slutligen utvärderar vi utvecklingsprocessen både tekniskt och organisatoriskt.

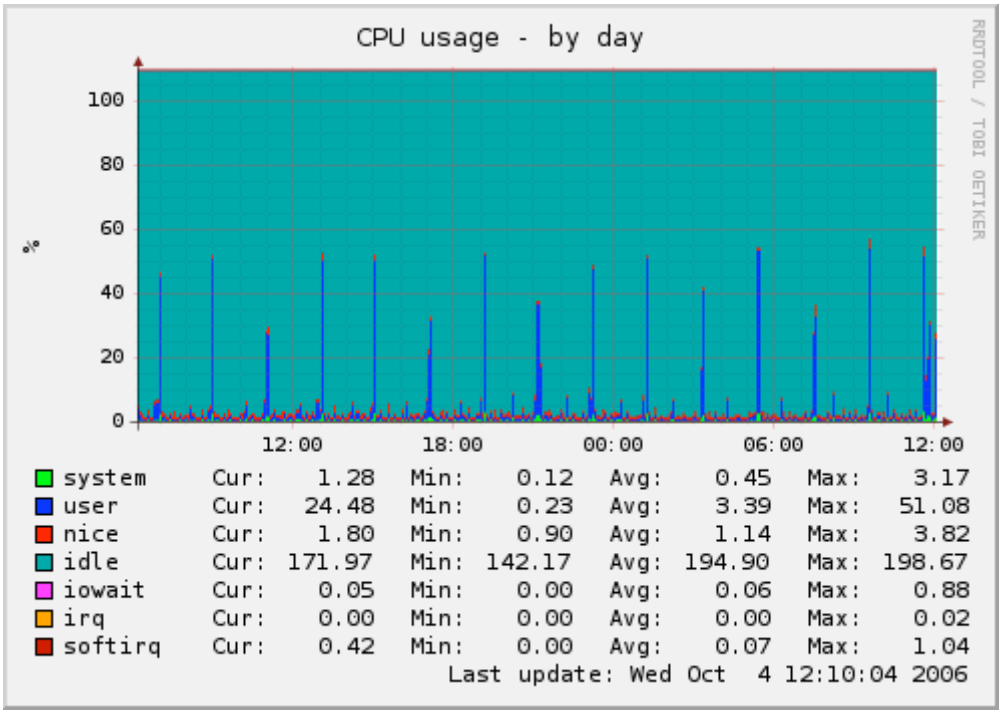
6.1 Servern

Att anpassa en server är en svår uppgift, av många anledningar. Det är mycket svårt att uppskatta hur stor den framtida trafiken kommer att bli, och det är också svårt att veta hur mycket resurser applikationen själv kommer att ta från servern. Det finns färdiga applikationer för s.k. load-testning, d.v.s. stresstest, till exempel Apache JMeter[56], men även dessa har sina begränsningar. Av dessa anledningar valde vi att helt enkelt att låta servern testas i skarpt läge. Vi visste att framgången inte kommer över en natt, därför hade vi både tid och utrymme att anpassa inställningar och liknande allt eftersom. Vi väntade också medvetet med att ta kontakt med media och liknande innan vi var säkra på att konfigurationen fungerade bra.

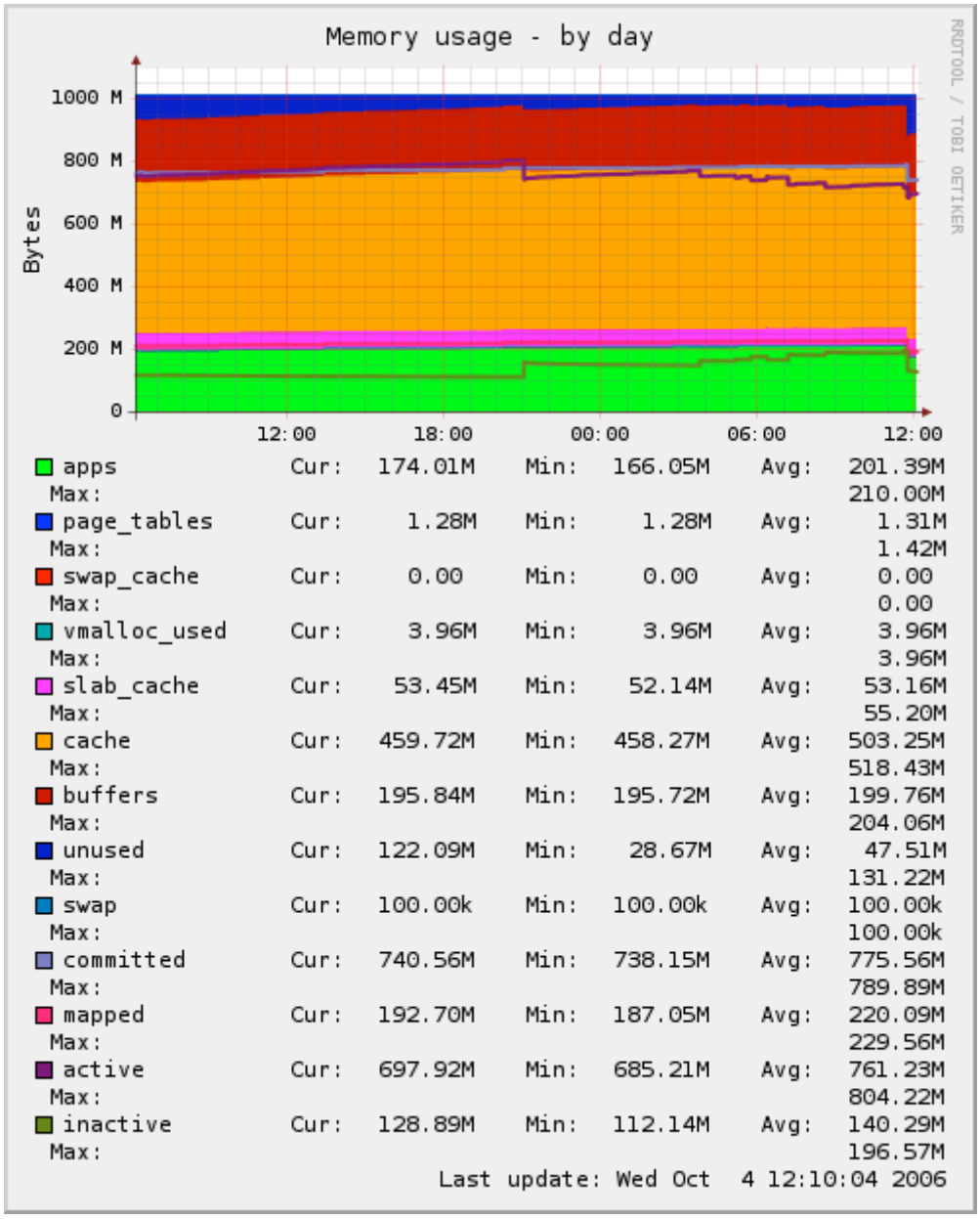
För att hålla koll på hur servern fungerar installeras tidigt ett diagnostikpaket kallat ”Munin”[57] på servern. Det består av ett program som körs var femte minut på servern och då skapar ett antal diagram över olika aspekter på prestandan. Dessa kan sedan nås via ett enkelt webgränssnitt, som säkrades för obehöriga användare genom att lägga den bakom en inloggning via Apache.

Det som var mest intressant i början var dels användningen av processorkraft (figur 5), dels minnesutnyttjandet (figur 6).

[56] Apache Jmeter, http://jakarta.apache.org/jmeter/, 2006-09-10
[57] Munin, http://munin.projects.linpro.no/, 2006-09-10



Figur 5 - Processoranvändning på servern



Figur 6 - Minnesanvändningen på servern.

Som vi ser i dessa diagram så utnyttjas processorn knappt alls under den absoluta majoriteten av tiden. Den enda gången den utnyttjas ordentligt är när oddsen från spelbolagen uppdateras varannan timme. Dock ansågs inte heller detta som ett problem eftersom webbservern alltid prioriterar att serva sidorna till användarna före att uppdatera oddsen. I slutet av den här uppsatsens skrivande köptes ännu en server in till Oddsnews, för att förbereda för en expansion. Planen är då att flytta oddsextraktorerna samt databasen till en server, och låta webbservern ligga kvar på den gamla. På det sättet får vi en server som är helt dedikerad till att serva sidor till användarna, vilket kommer att skala bättre den dagen detta blir nödvändigt.

Minnesanvändningdiagrammet ser kanske mer oroande ut vid en första anblick. Servern har 1gb internminne, och vet man inte något om minneshanteringen hos Linux kan det verka som att det faktiskt används till bristningsgränsen. En överanvändning av minnet skulle leda till att datorn började ”swappa” det vill säga använda hårddisken som

avlastningplats för internminnet, vilket är något som man absolut vill undvika, eftersom det avsevärt skulle slöa ner systemet.

Anledning till att diagrammet ovan inte är oroande är den stora delen av minnet som står tilldelat som ”cache”-minne. Linux ser till att använda allt tillgängligt internminne på ett så effektivt sätt som möjligt, och i cachen lagras ofta de mest använda områdena på hårddisken. Cachen växer och krymper automatiskt beroende på hur mycket minne som finns ledigt när applikationer tagit sitt.[58] Detta betyder att ett fullt utnyttjat internminne på ett Linuxsystem inte är ett tecken på något problem, utan helt enkelt att alla resurser utnyttjas så effektivt som möjligt.

Tillgänglighet är ett annat krav som är absolut avgörande för en server. Tjänsten måste finnas uppe så nära 100 % av tiden som det bara är möjligt. Även här har lösningen visat sig fungera över förväntan, sedan servern startades i skarpt läge den 19 juli 2006 har nedtiden enbart varit drygt en timme, och denna nertid berodde på ett misstag orsakat av den mänskliga faktorn. Det är dock svårt att extrapolera denna statistik till framtiden när servern kommer att få arbeta under en högre belastning, men baserat på erfarenheter från andra liknande webplatser är Linux och Apache en av de mest stabila och pålitliga plattformar för webblösningar som finns.

En annan viktig faktor, som faktiskt var helt avgörande för Oddsnews, är kostnaden för hela lösningen. Genom att enbart förlita oss på Open Source-lösningar, både för utveckling av projektet samt servern, har företaget inte behövt lägga ner en enda krona i mjukvarukostnader. Nackdelen med detta är i huvudsak avsaknaden av support som i många fall skulle ha följt med en kommersiell lösning, och den fragmentering som ibland karaktäriserar Open Source-lösningar. Det finns ofta många olika varianter av verktyg för att lösa samma problem, och för en ovan utvecklare kan det vara svårt att välja mellan dessa. Detta är givetvis både en styrka och en svaghet, men det är viktigt att vara medveten om detta innan man ger sig på ett projekt baserat på Open Source-lösningar. Inom Oddsnews har den nödvändiga kompetensen inom de olika områdena hela tiden funnits, vilket har gjort att Open Source har varit en självklar lösning för företaget.

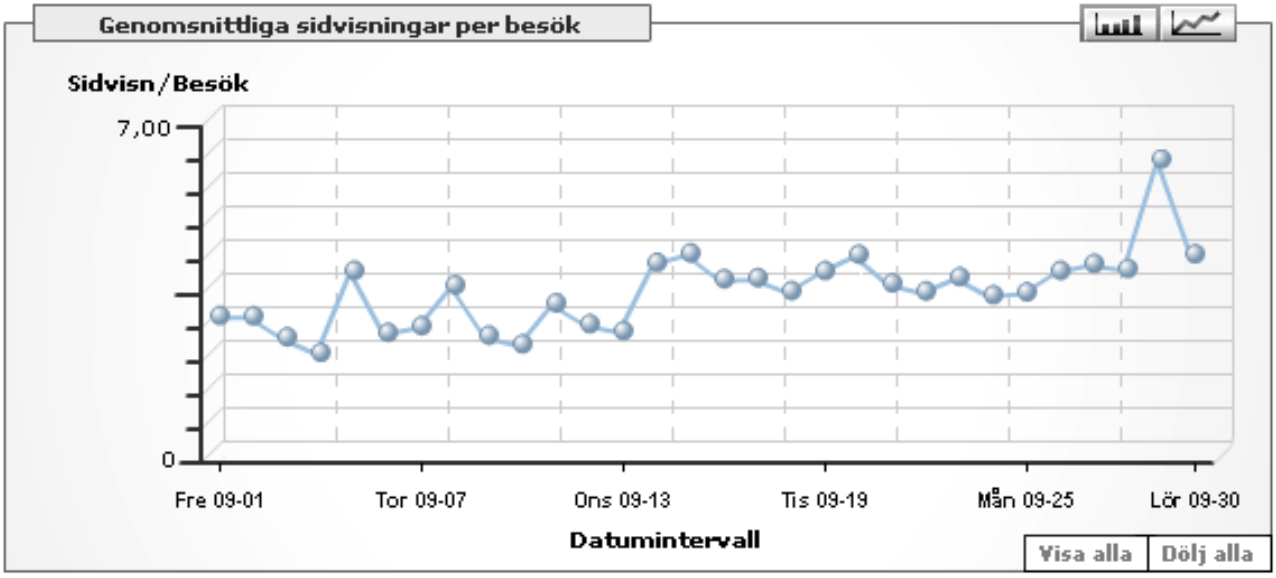
6.2 Användarstudie

Det slutgiltiga beviset på huruvida en utvecklingsprocess varit framgångsrik eller inte är om användarna av produkten är nöjda med resultatet. Beroende på hur förutsättningarna ser ut finns det olika sätt att mäta kundnöjdhet. I projekt med en väl definierad slutkund är det lättare att göra en sådan utvärdering, vilken då ofta kan göras i mer kvalitativa termer. I fallet Oddsnews.com är dock detta svårare – dels på grund av det stora antalet besökare samt den svaga kontakt som finns mellan utvecklarna och användarna. För att komma förbi detta problem kan man antingen försöka få en starkare kontakt med användarna – exempelvis genom webbenkäter eller genom att betala användarna för tester i en kontrollerad miljö. Detta är dock dyrt och det är inte säkert att man får ett representativt urval. En alternativ lösning för utvärdering, vilken valts, är att i kvantitativa termer försöka uppskatta webbsidans attraktionskraft. Genom att följa hur statistiken förändrats när ändringar på sidan genomförts har man försökt dra slutsatser om vilket genomslag förändringarna har fått. Nackdelen med detta är att det kan vara

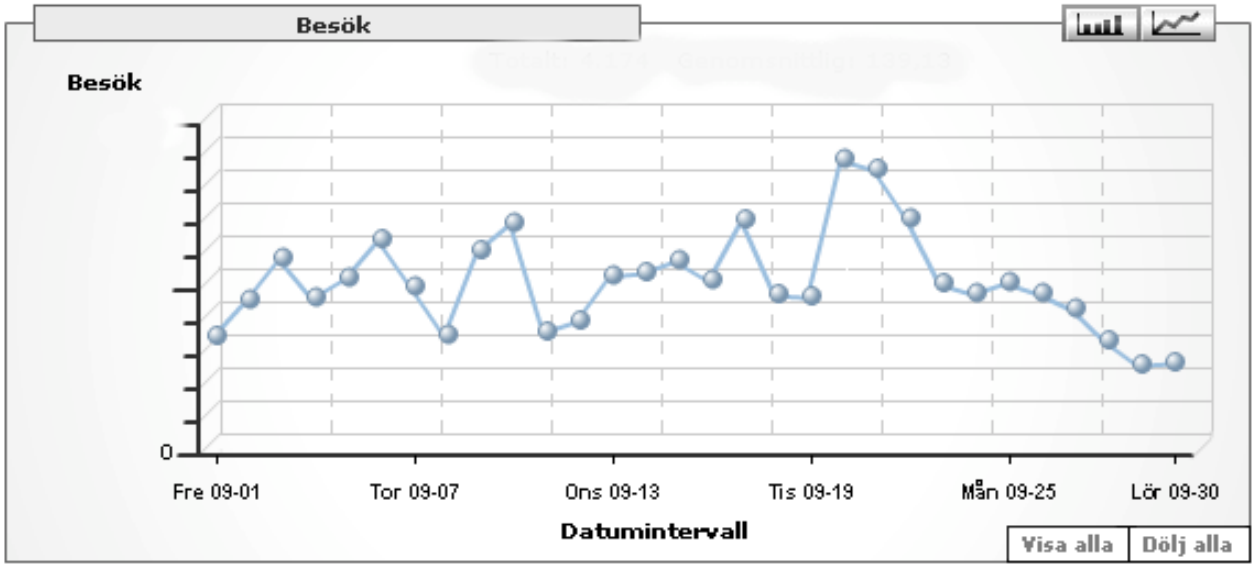
[58] http://www.faqs.org/docs/linux_admin/buffer-cache.html

svårt att dra tillbaka funktioner som inte fungerar bra när de redan har publicerats på sidan – eftersom personer har vant sig med sidans utseende och funktionalitet. Dessutom kan det vara svårt att verkligen kunna dra korrekta slutsatser eftersom man exempelvis inte har ett konstant antal besökare från vecka till vecka. Efter önskemål från uppdragsgivarna har vi dolt det absoluta antalet besökare och visar istället på trender (se figur 8). Detta innebär dock inte att även det absoluta antalet besökare är en viktig parameter att väga in i en utvärdering av en webbsidas attraktionskraft.

Som många sajtägare har erfarit är antalet besök ensam ingen bra indikator på framgången för en webbplats. Anledningen till det är att man lätt kan se tillfälliga ökningar av besökarantalet – som snarare beror på yttre omständigheter, som exempelvis slumpmässiga sökord hos sökmotorer eller tillfälliga ingående länkar. Detta leder till en tillfällig ökning av antalet besök men dessa besök är mycket korta och besökarna återkommer sällan. Målet med webbsidan är snarare att skapa ingående länkar och sökord som är relevanta och leder intresserade besökare från rätt målgrupp att delta i webbsidan på ett mer aktivt sätt. Vi anser därför att en bättre indikation för att mäta detta är antalet sidvisningar per besökare (se figur 7). På det sättet får man en bättre bild av hur webbsidan lyckas locka besökare att stanna kvar och använda sidan – något vi anser vara centralt för att bygga en långsiktig verksamhet.

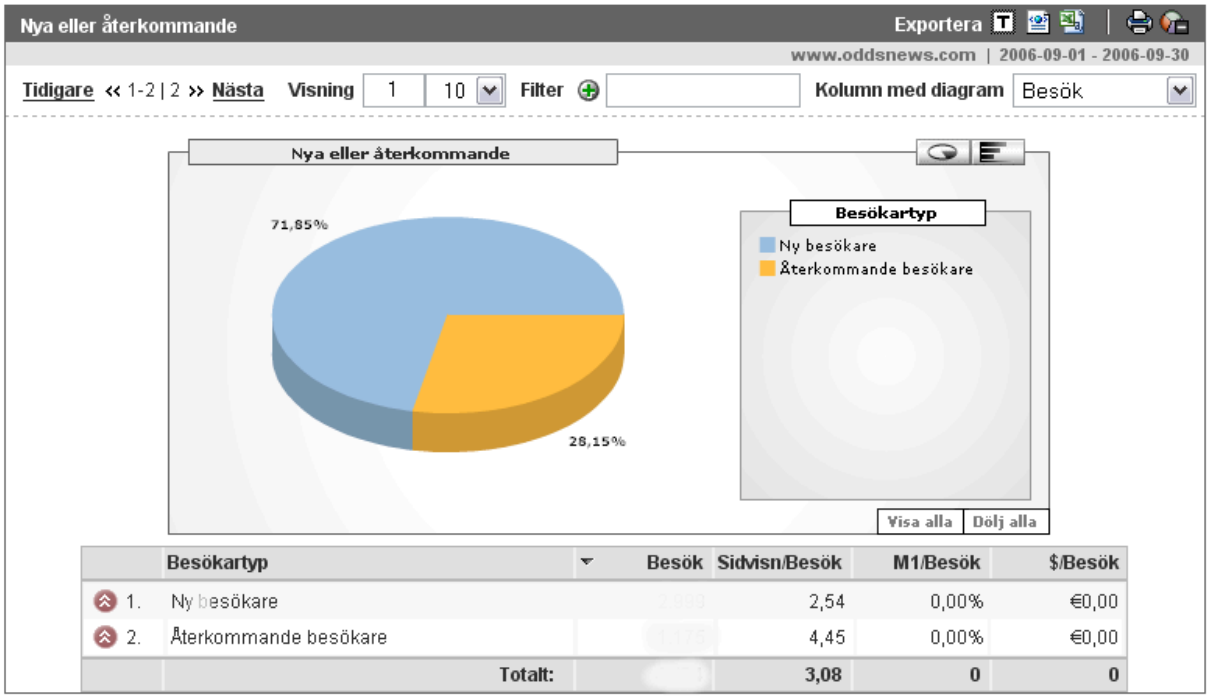


Figur 7 - Genomsnittliga sidvisningar per besök på Oddsnews.com.



Figur 8 - Antalet besök på Oddsnews.com.

Dessa två diagram visar antalet besök samt antalet sidvisningar / besök hos Oddsnews i september 2006. Tillsammans ger de en fingervisning om hur webbplatsen utvecklas. I mitten av månaden, närmare bestämt 13 september lanserade vi en ny version av Oddsnews. Nya funktioner i den var främst en förenklad navigation samt införandet av travoddsjämförelser. Ser vi till antalet sidvisningar per besökare efter denna tidpunkt så ökade dessa och höll sig konstant på en högre nivå än tidigare, samtidigt som antalet besökare höll sig någorlunda konstant. Av detta drar vi slutsatsen att den nya navigationen fyllde sitt syfte genom att locka besökarna att stanna längre på sidan. I diagrammen kan vi också se att vi framåt slutet av månaden har färre besökare samt en ytterligare ökning av antalet sidvisningar per besök. Detta är inte särskilt överraskande. I genomsnitt tenderar återkommande besökare att visa betydligt fler sidor per besök (se figur 7) – vilket medför att ett vikande antal nya besökare samtidigt leder till en ökning av antalet sidvisningar/besök. Detta skulle kunna vara en invändning mot att använda antalet sidvisningar/besök för att utvärdera attraktionskraften. Vad man alltså måste göra är att också titta på hur stor andel av besökarna som är nya besökare (se figur 9). En bättre variabel skulle alltså kunna vara antalet sidvisningar/ny besökare. Innan ändringen av designen (1-13 september) hade sidan 2,14 sidvisningar per ny besökare och efter ändringarna hade vi 2,85. Vi kan alltså se en ökning även här. Dock är den inte lika stor som ökningar för de återkommande besökarna vilken ökat från 3,37 till 5,06. Ändringen har alltså fått besökarna som återkommande använder sidan att använda den ännu mer – vilket får anses vara ett bra betyg för uppdateringen.



Figur 9 - Nya och återkommande besökare på Oddsnews.com.

7. Helhetsutvärdering

Som nämndes i inledningen är syftet med denna rapport att analysera de olika delar som en utvecklingsprocess består av och se hur dessa påverkar den slutgiltiga produkten. Historiskt har många projekt med tekniskt bra lösningar ändå resulterat i misslyckanden. Som breda ingenjörer ville vi bidra med ett vidare perspektiv under utvecklingsprocessen. Frågan är alltså om man kan dra några slutsatser om hela utvecklingsprocessen utifrån det vi redovisat under varje del ovan. Slutsatserna nedan blir naturligtvis tentativa eftersom utvecklingsprojektet fortfarande är i en utvecklingsfas. Dock kan vi redan nu se några tydliga mönster.

Tydligast är att ett projekt som Oddsnews.com har en hög risk att misslyckas. Därför har många av de strategier som valts under utvecklingen syftat till att minska denna risk – exempelvis genom att förenkla förändringar sent i utvecklingsskedet. Spring och kanske mer specifikt dependency-injection har visat sig bidra till en modulär arkitektur som gör det lätt att göra stora ändringar i koden utan att behöva ändra grundläggande delar av applikationens struktur. Detta gör det också lätt att utveckla ny funktionalitet allteftersom behovet framstår. Därför kan vi se att den stora användningen av ramverk bestående öppen källkod har varit en framgångsrik teknisk strategi. Något vi också märkt under projektets gång har varit det faktum att valet av tekniska lösningar även har ändrat de organisatoriska förutsättningarna som exempelvis utvecklingsprocesserna. Detta ses kanske tydligast i hur valet av utvecklingsmiljö med integrerat versionshanteringssystem har underlättat förutsättningarna för den geografiskt splittrade utvecklingsgrupp som projektet bestått av. Dock kunde detta även märkas i något så fundamentalt som valet av designarkitektur. Applikationens modulära MVC-design har möjliggjort för de olika utvecklarna att arbeta med olika oberoende delar av applikationen parallellt vilket har underlättat utvecklingsarbetet betydligt. Här har även förskrivna enhetstester – vilket sett till att diverse omskrivningar av olika delarna av applikationen inte har påverkat applikationen som helhet. Man kan alltså se en växelverkan mellan projektets tekniska utvecklingsarbete och dess organisatoriska förutsättningar.

En agil utvecklingsmetod ser vi nästintill som ett måste för att projekt som är så beroende av att dra besökare. Vi har haft begränsade möjligheter att skapa och använda ”Use cases”, mycket beroende på problematiken i att få kontakt med användarna. Trots detta har vi hela tiden lyssnat på vänner och bekanta som använt sidan samt uppmuntrat användarna att posta förbättringsförslag i Oddsnews forum.

Utan en flexibel organisation och kodbas som snabbt kan anpassas efter vad användarna efterfrågar är det mycket svårt att lyckas på en så trendkänslig marknad som Internetmarknaden ändå är. Där skulle en tung utvecklingsprocess ha dragit ner produktiviteten avsevärt, förmodligen skulle även arbetslusten hos utvecklarna minskat betydligt. Med de snabba iterationer som använts under utvecklingsprocessen har känslan varit att projektet ständigt rört sig framåt. Den testdrivna utvecklingen har, särskilt vid utvecklingen av oddsextraktorerna, visat sig mycket effektiv för att minska felfrekvensen i koden, men det har även visat sig snabba upp hela utvecklingsprocessen. Genom att ha automattester som kan köras på någon sekund slipper programmeraren att starta servern och vänta på att den ska initialiseras, vilket kan ta uppemot en halv minut i vissa fall. Det finns dock fortfarande mycket att utveckla på testfronten, idealt så skulle alla delar av applikationen ingå i testspecifikationen, medan den nu enbart

innefattar klasser som går att testa utan att köras i en Servlet container. Där kan testverktyg som Cactus, DBUnit med flera vara lämpliga att utvärdera.

En fälla det är lätt att hamna i inom ramarna för ett utvecklingsprojekt som detta, även om man är medveten om den, är att bli lockad av ett intresse för själva tekniken och därmed förlora fokus på helheten. Vad vi märkt är att vårt deltagande i utvecklingsprocessen på många sätt gett bidragit med nya infallsvinklar till projektet – dock kan det vara farligt att se en sts-ingenjör som en ersättning för andra kompetenser som är nödvändigt i ett utvecklingsprojekt av denna typ (exempelvis ekonomer eller informatörer). Ytterligare ett problem – som delvis kanske är oundvikligt i ett litet företag – är att enskilda personers kompetens görs oersättlig. Detta gör dessa personer till mänskliga så kallade single-point-of-failures utan vilka projektet avstannar. Därför ser vi att det i nuläget är viktigt att försöka sprida de olika kompetenserna inom företaget.

Bristen på kapital har visat sig vara ett överkomligt problem på den tekniska sidan. Att sätta upp en relativt kraftfull servermiljö är idag inte särskilt dyrt och Open source-lösningar har bidragit till att minska utvecklingskostnaderna avsevärt. Dock har de knappa finansiella resurserna varit ett större problem för att få ett marknadsmässigt genomslag på en marknad där det är väldigt viktigt att bygga ett varumärke. Vi kan även se en tendens till att de knappa resurserna på sikt urholkar möjligheterna att upprätthålla en snabb utvecklingstakt av projektet – något som ju från början upplevts som ett viktigt mål. Detta har även medfört en viss ryckighet i processen, eftersom ingen av personerna bakom Oddsnews helt har kunnat fokusera på uppgiften.

Tydligt är också att det krävs en långsiktighet och ett personligt engagemang i verksamheten även om den inte leder till en direkt framgång. Många lockas in i Internetbranschen av en vilja att tjäna snabba pengar, men i realiteten ser verkligheten annorlunda ut i nästan alla fall. Det är mycket svårt att på förhand definiera en strategi för att nå lönsamhet i en sådan osäker bransch och strategier som tidigare fungerat för andra entreprenörer behöver inte med nödvändighet fungera igen. Vad vi kan se har det tidiga utvecklingsskedet av Oddsnews till stora delar lyckats, speciellt kanske i teknisk bemärkelse. Dock har företaget en lång väg till lönsamhet och att det enligt oss i framtiden blir ännu viktigare att skapa strategier för att koppla användarnas behov till den tekniska lösningen och utvecklingsprocessen till den bredare affärsstrategin.

7.1 Avslutande personliga reflektioner

Att arbeta inom ett litet och nytt utvecklingsdrivet företag som Oddsnews Ltd har öppnat upp många nya perspektiv för oss som tvärvetenskapliga civilingenjörer. Därför tänkte vi ägna detta sista avsnitt åt en mer reflexiv utvärdering av arbetet med Oddsnews.com.

Dessutom har vi i projektet fått förhålla oss till en mer organisatorisk problematik. Exempelvis har vi konkret känt av behovet av att undvika att ryckas med i ett projekt och att därigenom tappa ett kritiskt förhållningssätt till produkten man utvecklar. Viktigast har kanske varit insikten om att ett konkret projekt inom ett företag skiljer sig mycket från den teori vi tagit del av från universitet. Inom ramen för detta projekt har vi haft förmånen att kombinera en vetenskaplig analys med möjligheten att lära oss av konkret praktiskt arbete – något som vi ser som kärnan i att vara civilingenjör. Detta är något som vi möjligtvis hade sett mer av tidigare under vår universitetsutbildning. Dock

inser vi att det är svårt att införliva en verklig praktiskt arbetserfarenhet inom ramen för en teoretisk universitetsutbildning. I utvecklingens inledningsskede uppfattade vi stundtals vad vi skulle kunna karaktärisera som en kulturkrock mellan vår vana att ta in kunskap på ett teoretiskt plan och de yrkesverksamma ingenjörernas mer praktiska erfarenhet. Kanske kan denna typ av praktiska examensarbeten ändå fungera som en brygga gentemot ett framtida arbetsliv.

Detta halvår vi har deltagit i utvecklingen hos Oddsnews Ltd har alltså varit både intressant och lärorikt och vi önskar projektet lycka till inför framtiden.

Referenslista

Tryckt Litteratur

Couloris, G. et.al (2001), *"Distributed Systems – Concepts and Design",* 3rd Edition*,* Addison-Wesley, Pearson Education, England

Gamma, et.al, (1994) "*Design Patterns – Elements of reusable Object-oriented software*

Gibbs, R. Dennis (2006), *Project Management with the IBM® Rational Unified Process®: Lessons from the Trenches*, IBM Press, USA

Stepanek, G. (2006), *Software project Secrets – Why software projects fail*, Apress, USA

Mahemoff , M (2006), *Ajax Design Patterns*, O'Reilly

Johnson R et.al.(2005) *"Proffessional Java development with the Spring framework*", John Wiley & Sons

Håkon Wium Lie & Bert Bos (1999), *Cascading Style Sheets: designing for the Web,* Addison Wesley

Artiklar

Ambler, S. "*Agile Model Driven Development (AMDD*) , http://www.agilemodeling.com/essays/amdd.htm (2006-09-06)

Royce, Winston (1970), "Managing the Development of Large Software Systems"*,* Proceedings of IEEE WESCON*, vol. 26, no. August, p. 1-9*.

Mcgee, David, (1999), "From Craftsmanship to Draftsmanship: Naval Architecture and the Three Traditions of Early Modern Design", Technology and Culture 40.2

Miller, R. (2002), "Demystifying extreme programming", http://www-128.ibm.com/developerworks/java/library/j-xp0813/, 2006-10-10

Sinan, A. (2002), *Understanding the Unified Process,* i Methods & Tools Spring 2002, http://www.martinig.ch/PDF/dmt0102.pdf

Parnas & Clements, *"A rational design process: How and why to fake it"* http://www.ece.utexas.edu/~perry/education/360F/fakeit.pdf, 2006-10-20

Johnson, R., "*Introduction to the Spring Framework",* http://www.theserverside.com/tt/articles/article.tss?l=SpringFramework, 2006-08-20

Philip Jonsson, *Beräkning av Sannolikheter för Utfall i Fotbollsmatcher – Oddsen på din sida*, Examensarbete D – Nationalekonomiska institutionen Uppsala Universitet VT 2006

Internetkällor

Bracha G. (2004), *"Generics in the Java programming Language",* http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf, 2006-10-25

Fowler, M. (2001), Beck, K., "*Interview with Kent Beck and Martin Fowler",*http://www.informit.com/articles/article.asp?p=20972&rl=1 (2006-09-23)

Highsmith, J., "*History of the Agile Manifesto*", http://www.agilemanifesto.org/history.html, 2006-09-25

Raymond, Eric S., *"A brief history of Hackerdom",* http://library.n0i.net/advocacy/hacker-history/, 2006-10-13

JUnit, Testing Resources for Extreme Programming", http://www.junit.org/index.htm, 2006-09-01

Wikipedia, Schematisk bild över Vattenfallsprocessen, http://en.wikipedia.org/wiki/Image:Waterfall_model.png, 2006-09-08

The Agile Manifesto, http://www.agilemanifesto.org/, 2006-10-12
Wikipedia, Extreme Programming, http://en.wikipedia.org/wiki/Extreme_Programming, 2006-10-10

Wikipedia, Plain Old Java Object, http://en.wikipedia.org/wiki/Plain_Old_Java_Object (2006-10-13)

Wikipedia, Really Simple Syndication, http://sv.wikipedia.org/wiki/RSS, 2006-09-20

The Sun Java Community Process Program, http://jcp.org/en/home/index, 2006-10-13

Java SE Techonologies, Java Database Connectitivity (JDBC) http://java.sun.com/javase/technologies/database.jsp , 2006-10-10

Johnson, R. Et.al (2006)., *"The Spring framework – Reference Documentation"*, http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html, 2006-10-16

Tomcat FAQ*,"Is Tomcat faster than serving static HTML pages than apache?,* http://tomcat.apache.org/faq/performance.html, 2006-10-10

Hibernate Annotations – Reference Guide, http://www.hibernate.org/hib_docs/annotations/reference/en/html_single/#entity-hibspec-collection, 2006-09-20

Apache Jmeter, http://jakarta.apache.org/jmeter/, 2006-09-10
Munin, http://munin.projects.linpro.no/, 2006-09-10

JAMA (A Java Matrix Package), http://math.nist.gov/javanumerics/jama/, 2006-09-20