Ruby on Rails eller Django?

En jämförelse av två ramverk för webbutveckling

MARTIN KARSBERG
och PÄR-JOHAN JÖNSSON

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2010

Ruby on Rails eller Django?

En jämförelse av två ramverk för webbutveckling

M A R T I N K A R S B E R G
o c h P Ä R - J O H A N J Ö N S S O N

Examensarbete i teknik och management om 15 högskolepoäng
vid Programmet för industriell ekonomi
Kungliga Tekniska Högskolan år 2010
Handledare på CSC var Christian Bogdan
Examinator var Stefan Arnborg

URL: www.csc.kth.se/utbildning/kandidatexjobb/teknikmanagement/2010/karsberg_martin_OCH_jonsson_par-johan_K10050.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Ruby on Rails eller Django? – En jämförelse av två ramverk för webbutveckling

Sammanfattning

Inom webbutveckling går trenden mot att fler och fler tjänster flyttar ut på Internet. Dessa tjänster blir också mer och mer avancerade med en stor del interaktivitet och dynamiskt innehåll.

Denna trend ställer nya krav på utvecklarna då mer avancerade funktioner efterfrågas samtidigt som den tillgängliga tiden är den samma eller kortare.

Ramverk för webbutveckling är verktyg som syftar till att underlätta för utvecklarna genom att erbjuda en plattform med färdiga byggstenar för de vanligaste funktionerna i kombination med en god struktur.

I denna artikel tittar vi närmare på två populära ramverk, Ruby on Rails och Django. Vi jämför dem på ett antal viktiga punkter för att belysa deras respektive fördelar samt nackdelar. Allt med syftet att förhoppningsvis kunna belysa några av de aspekter som är viktiga vid val av ramverk för webbutveckling.

Ruby on Rails or Django? – A comparison of two frameworks for web development

Abstract

The field of web development is experiencing a situation where more and more services make their way onto the Internet. These web services also tend to become more and more sophisticated implementing a large portion of interaction and dynamic content.

This kind of development requires developers to produce more advanced functionality in the same time frame or sometimes in a shorter time frame.

Web development frameworks are tools aimed at making life easier for developers by offering a platform with pre-coded building blocks. This helps solve a large portion of the most frequent programming tasks a developer faces and at the same time encourages a clean code design.

This paper takes a closer look at two popular frameworks, Ruby on Rails and Django, and compares them on a set of important focus areas in order to highlight their weaknesses and strengths. Hopefully this will shed some light on some of the key aspects to consider when it comes to choosing a web development framework.

Innehållsförteckning

Inledning

I detta avsnitt kommer vi kortfattat beskriva bakgrunden till det teknikområde som denna artikel handlar om. Även syfte, mål och avgränsning tas upp i detta avsnitt.

Inom webbutvecklingen har det skett en tydlig förändring de senaste åren där webbplatser har gått från att vara statiska elektroniska anslagstavlor till att bli mer och mer lika vanliga datorapplikationer med avancerade dynamiska funktioner. Detta har resulterat i nya effektivare verktyg för utvecklare. Dessa verktyg möjliggör utveckling av avancerade funktioner utan en markant ökning av tidsåtgång eller resurser. Verktygen är också tänkta att underlätta administrationen genom att de producerade webbplatserna får en tydlig och logisk struktur.

Bakgrund

Det var länge sedan det räckte med att enbart kunna programmera HTML för att utveckla en tilltalande webbplats. Dagens användare förväntar sig en högre grad av rikt innehåll, personlig anpassning och interaktivitet då många, i stor utsträckning, förlitar sig på dessa tjänster i sin vardag[1].

Numera går även utvecklingen mot att applikationer, som tidigare enbart kunde användas i den lokala skrivbordsmiljön på en dator, flyttar ut till en server på Internet varvid den lokala användaren interagerar med dessa genom sin vanliga webbläsare[2].

Samtidigt som att utvecklingen går mot mer avancerade webbtjänster befinner sig näringsliv och teknologi i en föränderlig miljö där förändringstakten hela tiden ökar. Detta medför att utvecklingsprojekt för ny mjukvara ofta råkar ut för förändringar under arbetets gång som ändrar förutsättningarna[3]. Dagens utvecklare behöver alltså utveckla flexibla webbapplikationer.

Nya metoder för att bedriva utveckling av mjukvara så som *agile development* och *extreme programming* kan delvis ses som ett resultat av denna utveckling.

I ljuset av detta resonemang är det förståeligt att utbudet av olika verktyg för att underlätta utvecklingsarbetet har ökat. Ramverk för att utveckla webbapplikationer är skapade för att förenkla utvecklingen av webbplatser. Det finns i dagsläget ett mycket stort antal ramverk baserade på de olika programmeringsspråk som kan användas för att bygga en webbapplikation. Det användargenererade uppslagsverket Wikipedia har för närvarande en lista med cirka 66 stycken olika ramverk baserade på de vanligaste programmeringsspråken[4]. Även om alla dessa ramverk inte är helt jämförbara med varandra eftersom vissa är väldigt specialiserade på en specifik tillämpning ger det ändå en uppfattning om dessa verktygs popularitet och spridning. Tanken med ett ramverk är att det ska underlätta för utvecklaren genom att erbjuda färdiga byggstenar för de vanligast förekommande funktionerna. Dessa byggstenar kan sättas ihop för att bygga en webbapplikation med en viss funktionalitet. Varje byggsten eller modul kan även oftast konfigureras i detalj av utvecklaren. På så sätt behöver inte en utvecklare uppfinna hjulet på nytt varje gång han eller hon behöver lägga till en funktion på sin webbapplikation. Förutom att erbjuda färdiga byggstenar för de vanligaste funktionerna erbjuder även de flesta ramverk en konsekvent och tydlig struktur för projektet vilket gör att testning, dokumentation samt

[1] San Murugesan, Web Engineering: Modelling and Implementing Web Applications, 2008, p. 9

[2] San Murugesan, Web Engineering: Modelling and Implementing Web Applications, 2008, p. 13

[3] Stefan Koch, Extreme Programming and Agile Processes in Software Engineering, Springer Berlin/Heidelberg, 2004, p. 86

[4]

underhåll av webbapplikationen underlättas betydligt. Denna struktur uppnås genom att ramverket anammar någon typ av designmönster för mjukvaruutveckling.

Med tanke på detta väldiga utbud i kombination med den stora mängd underliggande tekniker i form av webbservrar och databaser är det en utmaning för programmerare att välja rätt teknik och ramverk till ett utvecklingsprojekt. Vi vill med denna text ge en bild av en viss typ av verktyg som fått mycket uppmärksamhet den senaste tiden. Dessa verktyg brukar kallas *agile frameworks for web development*. Med detta menas ett ramverk som främjar produktivitet och minimerar den tid som utvecklaren behöver lägga på att skriva kod eller konfigurera projektet[5]. Genom att jämföra två populära ramverk hoppas vi kunna ge en översikt över hur ett modernt ramverk för webbutveckling är uppbyggt och på och vilket sätt det kan bidra till att göra utvecklingsprocessen mer flexibel och effektiv samtidigt som man erhåller en kod som är lätt att underhålla och uppdatera.

Problemformulering

Det ställs allt högre krav på utvecklare att snabbt och kostnadseffektivt skapa webbapplikationer. Ramverken Django och Ruby on Rails är skapade för att förenkla och effektivisera utveckling av webbaserade applikationer. Då dessa ramverk är komplexa och svåröverskådliga krävs en djupare analys av respektive ramverk för att få en bättre förståelse av vilka fördelar och brister respektive ramverk har.

Mål och Syfte

Syftet med vårt arbete är att jämföra två ramverk som används för utveckling av dynamiska webbplatser utifrån ett antal parametrar så som: skalbarhet, användarvänlighet, kodens omfattning, support med mera. Till denna uppgift har vi valt ramverken Django, som baseras på programmeringsspråket Python, samt Ruby on Rails som baseras på Ruby.

Avgränsningar

Båda de ramverk som finns med i denna undersökning är komplexa i sig och det finns otaliga tillägg att tillgå. Detta gör det svårt att göra en total kartläggning av dessa ramverk. Vi har för avsikt att begränsa vår undersökning till att utgöra en introducerande jämförelse som kan bidra till att läsaren får en övergripande bild över hur utvecklingsarbetet inom dessa ramverk går till.

En viktig diskussion kring dessa ramverk är dess skalbarhet, det vill säga hur väl det fungerar att driva och utveckla webbapplikationer när applikationen och antalet användare växer. Vi har i denna uppsats valt att undersöka denna aspekt utifrån först och främst ett teoretiskt perspektiv.

[5] Mehdi Jazayeri, Some Trends in Web Application Development, IEEE Computer Society, 2007

Metod

Som metod i vår undersökning har vi valt att dels undersöka ämnet utifrån ett teoretiskt perspektiv och dels utifrån praktiskt erfarenhet genom att själva skapa en applikation i respektive ramverk. En mer ingående beskrivning av metoden finns i detta avsnitt.

Innan vi inledde arbetet med att jämföra Ruby on Rails och Django valde vi att skapa oss en översiktlig bild av hur dynamiska webbplatser fungerar, samt vilka tekniker som ligger till grund för dessa. Detta för att båda författarna saknade större erfarenhet av det specifika området. Internet är naturligtvis den främsta källan för denna typ av information i form av hemsidor, artiklar, e-böcker, och elektroniska böcker. Företag eller organisation som står bakom de olika teknikerna erbjuder nästan alltid väldigt omfattande dokumentation på sina webbplatser i form av manualer, steg-för-steg beskrivningar, forum samt FAQ:s (*Frequently Asked Questions*).

Vår jämförelse av de två ramverken för webbutveckling består av två huvudsakliga delar. Dels har vi framställt en rent teoretisk del där vi utgått från den dokumentation som fanns tillgänglig främst från de två ramverkens respektive webbplatser. Utifrån denna information har vi sedan försökt besvara de frågor som vi ansåg ej var möjliga att finna svar på genom en praktisk metod. Till denna kategori hör vissa frågor om till exempel skalbarhet. Det var nämligen inte möjligt för oss att testa respektive ramverk i den omfattning som krävs för att utifrån praktiska erfarenheter kunna dra slutsatser om hur väl de fungerar i ett större sammanhang. Till exempel för att driva en större webbplats med flera tusen besökare dagligen.

I den andra delen av vår jämförelse valde vi att praktiskt testa de två ramverken genom att programmera en enkel bloggfunktion. Genom att utveckla en enkel blogg fick vi möjlighet att testa flera viktiga delar av respektive ramverk så som *databasinteraktion*, *hantering av layout* samt grundläggande hantering av *HTTP-requests* och *URL-bindningar*. Att praktiskt utveckla ett enkelt projekt gav oss också en uppfattning om hur interaktionen med respektive ramverk sker under själva utvecklingsprocessen och hur arbetsgången ser ut.

För att få en strukturerad och tydlig jämförelse så har vi valt att dela in jämförelsen i ett antal områden vilka kan anses vara relevanta. Vi fann det problematiskt att hitta litteratur som beskriver jämförelsemetoder för webbutvecklingsramverk, utbudet verkar vara begränsat. Det finns däremot ett större utbud när det gäller metoder för utvärdering av traditionell mjukvara. Dessa metoder är dock oftast inte direkt överförbara[6]

Följande sammanställning listar de delar som vi har valt att fokusera på i vår jämförelse.

- antal filer för en webbapplikation
- antal olika typer på filer (olika språk man måste lära sig)
- beroenden mellan filer
- skalbarhet
- struktur
- dokumentation för installation
- dokumentation för programmering
- community, epostlista

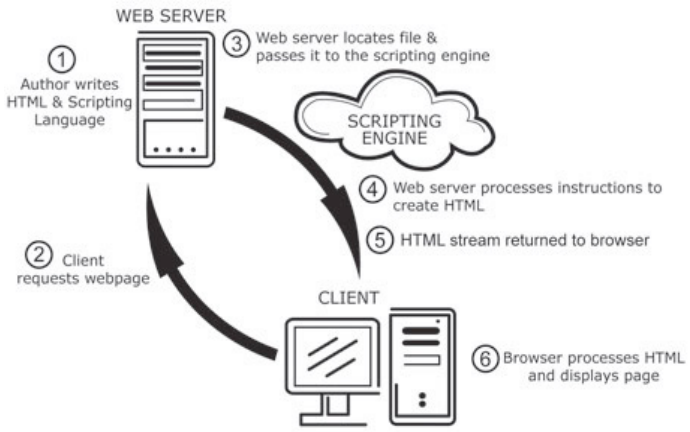
[6] José Ignacio Fernández-Villamor; Laura Díaz-Casillas; Carlos Á. Iglesias, A comparison model for agile web frameworks, Univ. Politécnica de Madrid, ACM, 2008

Teknik och bakgrund

Syftet med detta avsnitt är att ge en kortfattad beskrivning av de tekniska områden som är relevanta i vår jämförelse. Även relevanta designmönster beskrivs i detta avsnitt.

Dynamiska webbplatser

Det finns i princip två huvudsakliga tekniker för att presentera en webbsida som efterfrågas av en *klient* (webbläsare) från en *server*. I det ena fallet används en statisk princip där alla sidor som tillhör webbplatsen finns färdigkodade på servern. När klienten efterfrågar en viss sida är serverns enda uppgift att hitta rätt fil och skicka tillbaka den till klienten. I det andra fallet finns det inga färdiga webbsidor på webbservern. Istället genereras varje webbsida av ett så kallat script på servern innan den skickas tillbaka till användaren[7]. Fördelen med denna teknik är att klienten kan specificera vissa krav i sin förfrågan. Dessa krav tolkas sedan av scriptet och sidan genereras enligt klientens begäran. Detta är en förutsättning för att kunna erbjuda mer avancerade webbtjänster med användarinteraktion.



Figur 1: Schematisk bild som beskriver kommunikationen mellan server och klient av en dynamisk webbsida.

Kommunikationen mellan *klient* och *server* sker genom så kallade *HTTP-requests*. Dessa är meddelande som skickas mellan datorer över Internet vilka gör det möjligt för applikationer på olika platser att kommunicera med varandra enbart med hjälp av det standardiserade *HTTP-protokollet*[8]. En stor fördel är att de olika applikationerna inte behöver vara skrivna i samma programmeringsspråk. Bara de förstår *HTTP-requests* så kan de kommunicera. Vissa menar att denna teknik är absolut kritisk för utvecklingen av Internetbaserade applikationer[9].

Som vi har sett spelar webbservern en betydande roll för att en dynamisk webbplats ska fungera. Den fungerar som en exekveringsmiljö där det script som genererar det dynamiska innehållet exekveras.

[7] Alfred Wai-Sing Loo, Peer-to-Peer Computing, Springer London, 2007, p. 35

[8] Mark Pilgrim, Dive into Python 3, Apress, 2009, p. 225

[9] Daniel Cazzulino, Victor Garcia Aprea, James Greenwood and Chris Hart, Beginning Visual Web Programming in VB .NET, Apress, 2005, p. 344

Ramverk

Definitionen av *ramverk* (*framework*) varierar något inom området. Vi har här valt den definition som används av *Murray et. al.* i artikeln *An approach to specifying software frameworks*[10]:

> "*A Framework is a reusable design that requires software components to function.*"

En komponent definieras även i samma artikel som:

> "*A Component is a software unit (for example, a class, module, or function) or data unit that has a defined interface for which it (the component) provides an instantiation.*"

I artikeln belyser författarna alltså att för att ett ramverk ska fungera så måste utvecklare skapa de komponenter som ramverket kräver. Detta medför höga krav på utvecklare att sätta sig in och förstå ramverk som ofta är stora och komplexa i sin natur.

I dagsläget finns det en mängd olika ramverk tillgängliga för webbutveckling baserade på en stor grupp programmeringsspråk. Den största gruppen ramverk för webbutveckling är baserade på språket Java. Dessa ramverk erbjuder utvecklarna färdiga moduler som kan användas för att implementera vanligt förekommande funktionalitet på en dynamisk webbplats så som *autentisering av användare*, *databasåtkomst*, *URL-mappning*, *layout* med mera. För utvecklaren innebär detta att mer tid kan läggas på att utveckla den för webbplatsen unika funktionaliteten istället för att ödsla tid på grundfunktioner som redan implementerats många gånger tidigare.

De flesta *ramverk* främjar också användandet av olika designmönster i under utvecklingsarbetet av webbapplikationer. Detta underlättar utvecklingen och det framtida underhållet av webbplatsen. Just underhållet med utbyggnad av funktionalitet samt uppdatering av innehåll är något som anses vara en av de viktigare aspekterna att ta hänsyn till i val av ramverk samt under utvecklingsarbetet[11].

Övrig teknik

Både Django och Ruby on Rails använder sig av olika typer av tekniker för att leverera utlovad funktionalitet. I detta avsnitt har vi valt att belysa några av dessa.

HTTP server

Det är en *HTTP-server*, även kallad *webbserver*, som levererar sidorna till klienterna. Både Django och Ruby on Rails har möjlighet att använda sig av The Apache Software Foundations *HTTP-server*. Denna *open-source* server är världens mest använda *HTTP-server*[12] och används av otaliga sidor på internet.

Apaches *HTTP-server* har med åren blivit relativt komplex och kräver en del konfiguration för att användas. Den har dock visat sig fungera väldigt bra och projektet stöds av flera stora globala företag inklusive Microsoft och Google.

I Ruby on Rails inkluderas webbservern Mongrel. Mongrel är för applikationer skrivna i Ruby och kan endast hantera en förfrågan i taget vilket gör den oförmögen att hantera hög belastning av sig själv. För att klara högre belastning har Mongrel tidigare använts i *kluster* bakom en Apache webbserver som fördelar belastningen. Denna metod använde sig Twitter av tidigare för

[10] Murray et. al, An approach to specifying software frameworks

[11] http://www.tbray.org/ongoing/When/200x/2006/11/10/Comparing-Frameworks, inhämtat 2010-04-18

[12] http://httpd.apache.org/, inhämtat 2010-05-02

att hantera sin trafik på sin mikroblogg[13]. Nu för tiden rekommenderas istället Passenger[14], en modul till Apache webbserver. Denna modul kan även användas tillsammans med Ruby Enterprise Edition för högre prestanda[15]. Ruby Enterprise Edition är en alternativ tolk av Rubykod som kan användas istället för den vanliga tolken.

När Django installeras, installeras även en förenklad webbserver som endast är tänkt att användas under själva utvecklingsfasen. Denna webbserver klarar bara av, precis som Mongrel, att hantera en förfrågan åt gången. Detta gör den i princip oanvändbar när en Django applikation skall publiceras och börja användas på Internet. Istället är tanken att man övergår till en storskalig webbserver som Apache när webbapplikationen lanseras[16]. Webbservern från Apache kan i grunden inte tolka Python-kod men genom att använda en tilläggsmodul, som till exempel mod_wsgi, kan servern fungera som en vanlig exekveringsmiljö för Djangoapplikationen.

Databaser

I stort sett alla webbapplikationer kommunicerar med en databas för att lagra och hämta data. Ramverken i vår jämförelse hjälper programmeraren att konfigurera databasen och sköter kommunikation med dessa enligt *CRUD* funktionalitet. Förkortningen *CRUD* står för *Create, Read*, *Update* och *Delete*. När utvecklare använder sig av Ruby on Rails och Django så behöver de inte ta hänsyn till vilken specifik databastyp de använder sig av när de skriver komponenter som kommunicerar med databasen.

Designmönster

Designmönster kan beskrivas som mallar eller regler för hur utvecklare ska skriva sin kod. I detta avsnitt belyser vi några av de designmönster som främjas i Ruby on Rails och Django.

MVC

Ett populärt designmönster som både Ruby-on-Rails och Django tillämpar är *Model-View-Controller* eller förkortat *MVC*. Detta betyder att man separerar sin applikation i tre olika delar: *Model*, *View* samt *Controller*. Den grundläggande tanken med detta är att hålla kod som representerar data och logik (*Models*) separerat för kod som representerar de vyer (*Views*) man vill visa[17].

Model komponenter representerar data i applikationen och regler för hur data ska manipuleras. Detta brukar ibland kallas för *business logic* och enligt *MVC-designmönstret* bör denna del av programmet hållas separat ifrån *Views* för att det ska vara lättare att underhålla applikationen. Grundidén är att man inte ska behöva ändra något i *Model* bara för att man vill ändra sättet hur data visas i en *View* komponent.

View representerar gränssnittet mot användaren. *View*-komponenterna får data från *Controllers* och skapar sedan den vy som användaren ser. I Django och Ruby on Rails används inbäddad Python- respektive Rubykod i HTML-filer för att representera *View*-komponenterna.

[13] http://engineering.twitter.com/2010/03/unicorn-power.html, inhämtat 2010-04-28

[14] http://www.modrails.com/, inhämtat 2010-04-28

[15] http://www.rubyenterpriseedition.com/, inhämtat 2010-04-28

[16] http://djangobook.com/en/2.0/chapter12/, inhämtat 2010-04-12

[17] http://java.sun.com/blueprints/patterns/MVC-detailed.html, inhämtat 2010-04-28

Controller-komponentens uppgift är att hantera den information som kommer från användaren och kommunicerar med *Model*-komponenter för att hämta data. Datan delegeras sedan till *View*-komponenter.

DRY

Förkortningen *DRY* står för *Don't repeat yourself* och betyder "upprepa dig inte" på svenska. Denna princip brukar definieras som att: "Varje bit av kunskap måste ha en enda, otvetydig, auktoritär representation i ett system."[18]. Om denna princip följs fullt ut innebär det till exempel att om ett visst, för applikationen viktigt, värde måste justeras så ska detta enbart behöva göras på ett enda ställe. Därefter ska ändringen enligt logiska samband propagera ut till alla delar av applikationen som är beroende av den. Målet är att det ska gå snabbare att utveckla applikationer då man återanvänder kod i högre utsträckning. Detta medför även att risken för buggar minskar.

Representational State Transfer - REST

Representational State Transfer (*REST*) är definierat i en doktorsavhandling av Roy Fielding[19]. Följande avsnitt kommer belysa några av de principer som utgör *REST* och som är relevanta för den jämförelse som är avsikten med denna artikel. Vi vill dessutom uppmärksamma läsaren på att detta är en grov förenkling av begreppet *Representational State Transfer* och uttrycket *RESTful architecture* som även ofta används i detta sammanhang.

En viktig abstraktion som finns inom *REST* är resurser (*resources*). Dessa resurser kan vara allt från en HTML-fil, bilder till i stort sett vad som helst som kan nås genom en resursidentifierare (*resource identifier*)[20]. Med en resursidentifierare menas ofta en unik *URL* för att lokalisera resursen ifråga. Ett tydligt exempel på detta är att identifiera en bild genom *resursidentifieraren* - "*/bilder/17*", där "*17*" avser ett id på en specifik unik fil. Uttrycket *URI* (*uniform resource identifier*) används ofta för att identifiera en resurs. Till skillnad från *URL* så innehåller denna identifierare ingen information om vart man hittar resursen.

Vidare så kan en resurs ha olika typer av representationer. En klient kan begära en resurs, exempelvis en lista med kunder, som en textfil, *xml*-fil eller annat format som kan vara relevant för klienten. Detta underlättar om applikationen förväntas användas på flera olika typer av klienter.

En annan viktigt princip som tillämpas av *REST* är att kommunikation ska ske "*stateless*". Detta betyder bland annat att servern inte ska behöva kontakta sina klienter för att ta reda på vilken status de befinner sig i. Om detta var fallet så skulle arbetsbördan för servern snabbt bli allt för stor, då antalet klienter ökar. Applikationer som inte var programmerade enligt denna princip skulle då alltså inte skala väl[21].

Framtida trender

Utvecklingen av webbapplikationer förväntas fortsätta att utvecklas i snabb takt även i framtiden. Allt pekar på att behovet för företag att snabbt och billigt bygga ihop nya prototyper samt färdiga webbapplikationer kommer att öka i framtiden.

---

[18] http://c2.com/cgi/wiki?DontRepeatYourself, inhämtat 2010-04-14

[19] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, inhämtat 2010-03-24

[20] http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, s 88, inhämtat 2010-03-24

[21] http://www.infoq.com/articles/rest-introduction, inhämtat 2010-04-08

Några av de trender som är aktuella är *rapid application development*, en iterativ metod för att snabbt kunna utveckla flexibla prototyper och minska behovet av planering. Det finns också indikationer på att traditionella *CMS - Content Management Systems* som har använts för att underhålla webbplatser, börjar förändras och får utökad funktionalitet som mer liknar de ramverk som används för att skapa och underhålla webbapplikationer. Idag är huvuduppgiften för *CMS*-system inte endast att organisera innehåll utan de möjliggör också utveckling och underhåll av webbapplikationer.

Ruby on Rails

Ruby on Rails är ett ramverk skapat av David Heinmeier Hansson och bygger på programmeringsspråket Ruby. I detta avsnitt ger vi en överblick av Ruby on Rails och dess historia.

Programspråket Ruby

Ruby on Rails bygger som namnet antyder på programmeringsspråket Ruby. Ruby är skapat av Yukihiro Matsumoto och lånar mycket från andra programmeringsspråk som exempelvis Perl, Smalltalk, Ada, Lisp med flera. Ruby är ett objektorienterat interpreterande språk som först släpptes 1995. Att Ruby är interpreterande betyder att programmen måste köras i en tolk. År 1995 är för övrigt samma år då programmeringsspråket Java släpptes. Rubys skapare Matsumoto "*försökte göra Ruby naturligt, inte enkelt, på ett sätt som speglar livet*"[22]. Det var inte förrän ramverket Ruby on Rails släpptes som popularitet snabbt ökade för Ruby. En ytterligare anledning till den sena ökningen av popularitet anses vara att det saknades litteratur på engelska.

Här är en lista på några av de mest utmärkande dragen hos Ruby:

- Allt är ett *objekt* och alla *klasser* ärver från klassen *Object*.
- *Metoder* kan anropas med eller utan avslutande parenteser. *Metoder* kan även ha ovanliga tecken som "?" och "!" i sig.
- I Ruby är allt som inte är *nil* eller *false* sant, även värdet 0 samt tomma *listor*.

```ruby
# Say hi to everybody
def say_hi
  if @names.nil?
    puts "..."
  elsif @names.respond_to?("each")
    # @names is a list of some kind, iterate!
    @names.each do |name|
      puts "Hello #{name}!"
    end
  else
    puts "Hello #{@names}!"
  end
end
```

Figur 2: Kod skriven i programmeringsspråket Ruby

Historia

Ruby on Rails är en delprodukt från utvecklingen av den kommersiella webbapplikationen Basecamp[23] och är skapat av dansken David Heinmeier Hansson. Ruby on Rails har även använts för att utveckla den populära webbapplikationen Twitter[24]. På senare tid så har dock

[22] http://www.ruby-lang.org/, inhämtat 2010-05-01

[23] http://basecamphq.com/, inhämtat 2010-05-01

[24] http://www.twitter.com, inhämtat 2010-05-01

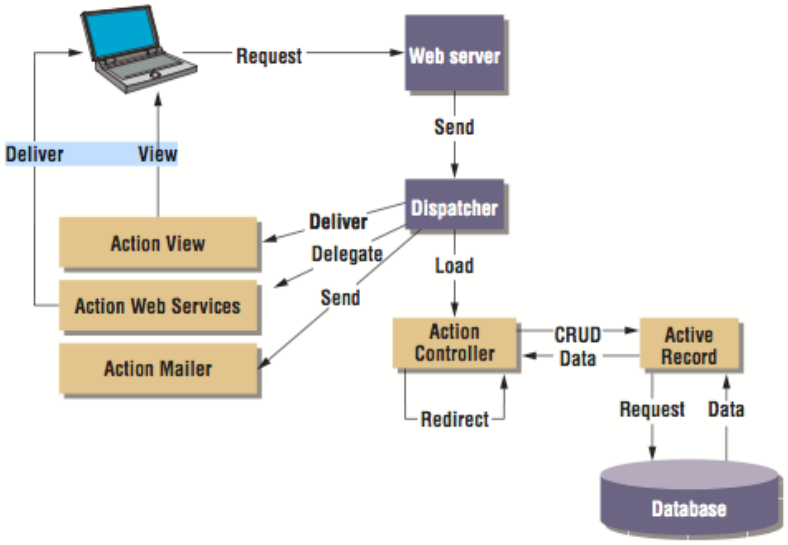
ramverket hamnat lite i blåsväder när anställda på företaget som utvecklar Twitter meddelade att de hade slutat att använda Ruby on Rails i vissa delar av sina applikationer då de behövde bättre prestanda[25]. Ramverket har dock visat att det fungerar väl och det finns många lyckade exempel på webbplatser som idag använder Ruby on Rails.

Uppbyggnad

Ruby on Rails består av sex stycken *komponenter*. Dessa finns namngivna i följande lista.

- Action Controller
- Action View
- Active Record
- Action Mailer
- Active Resource
- Railties
- Active Support

De tre viktigaste delarna är representerade av *Active Record*, *Action View* samt *Action Controller*. Dessa motsvarar *Model*, *View* respektive *Controller* i MVC designmönstret[26].



Figur 3: Diagrammet ovan visar hur komponenterna i Ruby on Rails kommunicerar med varandra.

Active Record

Active Record är den komponent som representerar modeller i MVC designmönstret. I Ruby on Rails manipuleras data genom att komponenten kommunicerar med en databas. Logiken för hur data får ändras implementeras här i enlighet med MVC designmönstret.

[25] http://www.radicalbehavior.com/5-question-interview-with-twitter-developer-alex-payne/, inhämtat 2010-04-28

[26] Ruby on Rails, Michael Bächle and Paul Kirchberg, 2007, IEEE Software

```ruby
class Post < ActiveRecord::Base
  validates_presence_of :name, :title
  validates_length_of :title, :minimum => 5
end
```

Figur 4: Ett exempel på en klass som ärver ifrån Active Record komponenten.

Action View

Action View är den komponent som representerar vyer i MVC *designmönstret*. I en Ruby on Rails applikation är detta HTML-filer med inbäddad Ruby kod.

För att tillämpa principen *Don't Repeat Yourself*, använder *Action View* något som kallas för *partials* eller *partial templates*. Detta kan närmast beskrivas som en form av mall. Dessa mallar representerar en del av det som ska visas. Detta gör att man inte behöver skriva samma kod för att exempelvis visa ett tabellhuvud för varje *View*.

```
<h1>New post</h1>

<% form_for(@post) do |f| %>
  <%= f.error_messages %>

  <p>
    <%= f.label :name %><br />
    <%= f.text_field :name %>
  </p>
  <p>
    <%= f.label :title %><br />
    <%= f.text_field :title %>
  </p>
  <p>
    <%= f.label :content %><br />
    <%= f.text_area :content %>
  </p>
  <p>
    <%= f.submit 'Create' %>
  </p>
<% end %>

<%= link_to 'Back', posts_path %>
```

Figur 5: Ett exempel av html-kod med inbäddad Ruby-kod.

Action Controller

Action Controller är den *komponent* som representerar *Controller* i MVC *designmönstret*. En *Controller* i Ruby on Rails har som uppgift att dirigera kommunikationen inom applikationen och levererar data som används av *Action View*.

Action Mailer

Denna komponent utgör ett ramverk för att ta emot och skicka e-post i Ruby on Rails. Mallar används för att skapa e-post som sedan skickas.

Active Resource

Det är komponenten *Active Resource* som möjliggör att Ruby on Rails kan tillämpa de principerna som finns inom *REST* för hur resurser ska hanteras. *Active Resource* har som funktion att kartlägga de resurser som finns inom applikationen.

Railties

Är den grundläggande komponenten som sammanbinder de olika ramverken som utgör Ruby on Rails. Används för att skapa och länka samman applikationer som är byggda i ramverket.

Active Support

Denna komponent utgör ett bibliotek med stödklasser för användaren samt den grundläggande koden i Ruby on Rails.

Underliggande tekniker

Databaserna SQLite, MySQL och PostgreSQL stöds direkt[27] och kräver relativt lite arbete av användaren för att användas inom applikationen. Följande lista visar de databaser som stöds enligt utvecklarna[28]:

- SQLite
- MySQL
- PostgreSQL
- IBM DB2
- Microsoft SQL Server
- Oracle
- Sybase
- Firebird

Utmärkande drag

Ruby on Rails använder sig av programmeringsfilosofi *Don't Repeat Yourself* och har funktionalitet som möjliggör att utvecklare kan använda sig av denna metod.

En annan viktigt filosofi som främjas i Ruby on Rails är *Convention Over Configuration*, eller "konvention före konfiguration". Alltså att ramverket förutsätter att programmeraren vill göra vissa saker utan att denne behöver ställa in exakta detaljer för hur dessa saker ska göras. Detta medför också att Ruby on Rails förutsätter en del saker som exempelvis hur filer döps etc. Detta kan göra att utvecklingsarbetet går snabbare men låser också användaren till att utveckla på ett visst förutbestämt sätt.

REST - Representational State Transfer, främjas inom Ruby on Rails, bland annat genom komponenten *Active Resource* och utvecklarna anser detta vara det bästa sättet att utveckla webbapplikationer på.

[27] http://guides.rubyonrails.org/, inhämtat 2010-04-20

[28] wiki.rubyonrails.org, inhämtat 2010-04-20

Django

Ramverket Django är utvecklat av The World Company och är baserat på programmeringsspråket Python. I detta avsnitt presenteras detta ramverk närmare.

Programspråket Python

Django är skrivet i programmeringsspråket Python vilket är ett interpreterat, dynamiskt, objektorienterat språk. Den första versionen av Python utvecklades i slutet av 1980-talet av en ABC-programmerare vid namn Guido van Rossum. Han ansåg att det saknades en del funktioner, framförallt automatisk felhantering, i de språk som fanns tillgängliga vid den tiden. Guido van Rossum är fortfarande delaktig i utvecklingen av Python genom stiftelsen "The Python Software Foundation"[29]. Under 2008 släpptes den senaste versionen 3.0.

Python jämförs ofta med andra interpreterade språk så som: Tcl, Perl, Ruby, Scheme eller Java och lämpar sig, precis som dessa språk, även som *scriptspråk*. Dock är användningsområdet brett och Python används i de flesta tillämpningar. Python har flera utmärkande egenskaper varav några är:

- Tydlig syntax
- Väl utbyggt standarbibliotek
- Lätt att lära
- Fri källkod

För att avgränsa kodblock behövs inga "måsvingar" (”{” och ”}”) eller dylikt. Istället används ett vanligt indrag för att skilja mellan olika kodblock. Kodrader i Python behöver heller inte avslutas med ett semikolon vilket annars är vanligt bland andra programmeringsspråk. Detta är två exempel på egenskaper som enligt utvecklare bidrar till att Python har en tydlig syntax.

I Pythons standardbibliotek finns inbyggt stöd för att genomföra en stor bredd av de uppgifter som kan förekomma i en modern applikation. Så som att packa upp komprimerade filer eller att hantera kommunikation med en webbserver.

Python har en väl utvecklad *community* med massor av användare som genom olika diskussionsforum kan ge svar på de flesta frågor. Dokumentationen för Python, som finns inbyggd i språket samt tillgänglig på nätet, är utvecklad med syftet att vara så lättfattlig som möjligt[30].

Python licensieras öppen källkod vilket gör att språket är fritt att använda och distribuera både för personliga samt kommersiella ändamål.

Historia

Django utvecklades av IT-avdelningen på The World Company - ett medieföretag med huvudsäte i Kansas, USA. The World Company ger ut ett antal dagstidningar som publiceras på Internet. Utvecklingen av Django genomfördes eftersom det fanns ett behov av en effektiv metod för att snabbt utveckla webbapplikationer för att klara journalisternas deadlines och publicera dessa på företagets webbplatser. Den första publika versionen av ramverket lanserades sommaren 2005 och är för närvarande uppe i version 1.1[31]. Eftersom Django bygger på flera

[29] http://www.python.org/doc/faq/general/, inhämtat 2009-04-15

[30] http://www.python.org/about/, inhämtat 2010-04-20

[31] http://docs.djangoproject.com/en/1.1/faq/general/, inhämtat 2010-04-18

tekniker som licensieras under öppen källkodslicens så som Apache, Python och PostgreSQL så valde utvecklarna att även licensiera Django under samma typ av licens. Numera drivs utvecklingen av Django av en internationell grupp av volontärer.

Uppbyggnad

Django använder sig av designmönstret *Model-View-Control* (*MVC*). Utvecklarna har dock valt att implementera designmönstret på ett lite annorlunda sätt. Till exempel menar de att en vy enbart beskriver den data som presenteras för användaren, inte nödvändigtvis hur den visas rent visuellt. Istället har varje Django applikation något som kallas en *Template* vilken är ansvarig för den visuella presentationen. Modellen är fortfarande en modell medan en Django applikation inte innehåller någon *Controller* i vanlig mening. Snarare menar utvecklarna att controllern är inbyggd i ramverkets underliggande funktionalitet. Djangos designmönster skulle alltså snarare kunna beskrivas som MTV, *Model-Template-View*[32]. I Djangos dokumentation separerar utvecklarna ramverket i fyra huvuddelar kallade: *Model layer*, *View layer*, *Template-layer* och *Forms*.

Model-layer

Till detta lager hör allt som har med den aktuella applikationens modeller att göra, så som databaskopplingar och de olika klassdefinitionerna. Här finns även de abstrakta klasser i ramverket som alla modeller ärver ifrån. Dessa klasser innehåller en mängd färdiga metoder för att utföra de vanligaste manipulationerna av data.

```
from django.db import models
from datetime import datetime

# Create your models here.

class Category(models.Model):
    name = models.CharField(max_length=60)

    def __unicode__(self):
        return self.name

class Post(models.Model):
    title = models.CharField(max_length=120)
    slug = models.SlugField(max_length=120, unique=True)
    body = models.TextField()
    published = models.DateTimeField(default=datetime.now)
    categories = models.ManyToManyField(Category)

    def __unicode__(self):
        return self.title

    class Meta:
        ordering = ['-published']
```

Figur 6: Ett exempel ur models.py.

View-Layer

I Djangos *View layer* återfinns all funktionalitet som rör projektets olika *View*-komponenter. Varje URL i ett Django-projekt har en koppling till en *View*. En *View* har i princip en huvudsaklig uppgift vilken är att ta emot en begäran och leverera ett svar i form av ett HTML-dokument, XML-dokument, bild eller liknande. Det finns ett stort antal möjligheter och det mesta är helt upp till utvecklaren.

[32] http://docs.djangoproject.com/en/1.1/faq/general/, Inhämtat 2010-04-18

Django har ett antal så kallade generiska *Views* inbyggda vilka kan användas av en utvecklare för att lösa de vanligast förekommande situationerna.

```
from django.shortcuts import render_to_response

def static_page(response, template):
    template = "%s.html" % (template)
    return render_to_response(template)
```

Figur 7: Ett exempel på en vy i Django

Template-layer

Djangos *Template layer* är tänkt att separera en webbplats layout från den individuella funktionaliteten hos de olika delarna. En *Template* är en HTML-fil som innehåller HTML-kod samt Djangos speciella *Templatespråk*. *Templatessystemet* använder ett eget taggspråk som används för att bädda in dynamiskt innehåll i HTML-filerna. Syntaxen påminner till viss del om Python-kod men är väldigt förenklad.

Forms

Den sista huvuddelen i Django är *Forms layer*. Detta kan ses som ett kodbibliotek som hanterar allt som har med forms och inmatning från användare att göra. Här finns ett stort antal fördefinierade typer av forms för att hantera olika former av inmatning utan att utvecklaren ska behöva bekymra sig för kontroll av denna.

Underliggande tekniker

Som har nämnts tidigare bygger Django, precis som alla andra ramverk, på ett antal underliggande tekniker. Den viktigaste är programmeringsspråket Python som hela ramverket är skrivet i. Python används genomgående i hela ramverket, även i till exempel konfigurationsfilerna, vilket gör att utvecklare ej behöver hoppa mellan olika språk när de utvecklar en applikation vilket annars är vanligt i många andra ramverk. Django är gjort för att köras mot en Apache webbserver. För att lagra data finns det i Django stöd för bland annat PostgreSQL och MySQL. För mindre projekt kan Django använda en databas kallad SQLite[33].

Utmärkande drag

Utvecklarna av Django har anammat ett antal vedertagna principer för programutveckling i designen av ramverket. Den mest framträdande av dessa är *Don´t Repeat Yourself* vilket bland annat återspeglas i Djangos *Template-system* där *Templates* kan ärva grundfunktionalitet från andra *Templates*.

[33] http://docs.djangoproject.com/en/1.1/intro/tutorial01/#intro-tutorial01, inhämtat 2010-03-27

Resultat

I denna del presenterar vi resultatet av vår jämförelse. Vi kommer utgå från de punkter vi presenterade i vår metod och titta på hur Django och Ruby on Rails hanterar dessa punkter i praktiken.

Filstruktur

Båda ramverken skapar en filstruktur för sina applikationer. Ruby on Rails har även ett kommando som heter *scaffold*, för att utvecklare snabbt ska kunna generera olika filer för ändamålet. Kommandot använder sig av *Convention over Configuration* designmönstret och passar därför inte alltid de önskemål utvecklaren har. Många utvecklare föredrar istället den kontroll som innebär om man skapar all kod själv och använder därför inte detta kommando nämnvärt[34].

Ruby on Rails

app/	här ligger själva applikationen
config/	
db/	
doc/	
lib/	
log/	
public/	publik katalog - här läggs bilder etc.
script/	
test/	testkod för applikationens klasser
tmp/	
vendor/	
Rakefile	
README	

Tabell 1: Tabelllen visar strukturen av ett Ruby on Rails program.

[34] http://guides.rubyonrails.org/getting_started.html#getting-up-and-running-quickly-with-scaffolding, inhämtat 2010-04-20

Innehållet i katalogen *app/*:

controllers/	
helpers/	
models/	
views/	

Tabell 2: Tabellen visar innehållet i katalogen apps/ (se tabell 1). Här syns tydligt uppdelningen enligt designmönstret MVC .

Django

Ett Djangoprojekt (en webbplats) och alla tillhörande resurser läggs oftast i ett gemensamt bibliotek, även om det inte finns några hinder för att även hämta in resurser från andra platser. Django skiljer på ett projekt och en applikation. Ett projekt utgör en samling av applikationer, mallar och statiskt innehåll, medan en applikation avser en återanvändbar delfunktion så som till exempel en kommentarsfunktion. I praktiken kan dock en applikation lika gärna utgöra en del av eller en hel webbplats. Det finns egentligen inga speciella krav på en applikation.

Varje applikation har en egen uppsättning av filer som representerar *Model* och *View*. Båda dessa är filer med Python-kod där *Model*-filen innehåller klasser som representerar applikationens olika modeller. *View*-filens uppgift är att generera ett *HTTP-respons* utifrån en så kallad *Template*. En *Template* är en html-fil som innehåller taggar i Djangos speciella *Templatespråk* vilka styr det dynamiska innehållet.

Andra viktiga filer i ett projekt eller applikation är url-filen samt *settings*-filen. I url-filen specificerar utvecklaren ett antal reguljära uttryck (*regular expressions*) vilka har till uppgift att tolka de url-adresser som tas emot från en *HTTP-request* och omvandla dessa till funktionsanrop till rätt del av programmet. Detta upplägg möjliggör en väldigt enkel och tydlig url-struktur vars utseende helt och hållet kan skräddarsys av utvecklaren. På så sätt undviks de orimligt långa och komplicerade url-adresser som är vanliga i PHP- eller ASP-miljöer. *Settings*-filen innehåller, som namnet antyder, en samling inställningar för det specifika projektet eller applikationen.

I tabellen nedan redovisas de viktigaste delarna av ett Djangoprojekt.

mysite/ (godtyckligt namn)	Huvudkatalog för projektet.
templates/	Projektets samling av templates.
urls.py	Projektets url-kopplingar.
settings.py	Inställningar för projekt/app.
views.py	Kopplar templates till en request.
models.py	Python klasser för modellerna.

Tabell 3: Filstrukturen för ramverket Django

Filtyper

Både Active Record- och Action Controller-komponenter i Ruby on Rails använder sig av Ruby-filer med ändelsen ".rb". Action View komponenter använder sig dock av vanliga html-filer med inbäddad Ruby-kod. Dessa filer har ändelsen ”.html.erb” för att indikera html och "embedded ruby", alltså inbäddad Ruby-kod.

Django använder Python-filer med ändelsen ”.py” för de flesta filer förutom *Templates* som är HTML-filer med inbäddade taggar i Djangos *Template-språk*, vilket till stor del liknar Python-kod med vissa undantag.

Vår testapplikation hade 120 filer när vi skapat den i Ruby on Rails. Detta ska jämföras med Django som hade 54 stycken filer.

Programspråk

Skaparen till Ruby har sagt i en intervju att en av anledningarna att han skapade språket var att han inte tyckte att Python var tillräckligt objektorienterat, utan en hybrid av procedurell- och objektorienterad art[35]. Han menade också att Python inte känns som ett *skriptspråk*. Detta till trots så har språken många likheter. Båda är tolkande (interpreterande) språk, det vill säga, de körs genom en mjukvara - en tolk.

Både Ruby och Python använder sig av "dynamisk typecasting". Detta betyder att objekt typbestäms under tiden koden körs. Det finns både fördelar och nackdelar med denna typ av programmeringssätt som kan skapa problem vilka är svåra att upptäcka vid utveckling av applikationer.

Python och Ruby liknar varandra på många sätt. Vi fann att de skillnader som finns mellan språken inte skapar några större fördelar eller nackdelar mellan dem. Här anser vi att det är viktigare att utvecklarna är erfarna inom respektive språk.

Designmönster

Det finns flera designmönster som de båda ramverken Ruby on Rails och Django använder sig av för att möjliggöra en effektiv utveckling av applikation. Vi vill i detta avsnitt visa hur de båda ramverken implementerade de filosofier som dessa designmönster står för.

[35] http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html, inhämtat 2010-03-21

MVC

Både Ruby on Rails och Django implementerar som tidigare nämt designmönstret MVC.

Model

I Ruby on Rails fungerar komponenten *Active Record* som *Model*. Komponenten representeras av klassen *ActiveRecord* som alla *Model*-komponenter i Ruby on Rails ärver ifrån. Funktioner som databas-oberoende, CRUD-funktionalitet med mera finns inbyggt i ramverket.

I Django definieras alla *Model*-komponenter, tillhörande en viss applikation, som klasser i en Python-fil. Varje sådan klass ärver från Djangos egen abstrakta Model-klass vilken innehåller ett ramverk för att utföra de vanligaste operationerna. Generellt sätt motsvarar en modellklass mot en tabell i databasen. Samtidigt som modellen skapas så får utvecklarna även tillgång till Djangos API för databaskommunikation abstraherar de vanligaste operationerna så som läsa, uppdatera och skriva. Fördelen med detta är att all interaktion med databasen kan skrivas i vanlig Python-kod och därigenom elimineras behovet av att inbädda SQL-satser, alltså kan utvecklaren fokusera på ett programmeringsspråk.

Controller

I Django finns inget som uttryckligen heter eller kallas för Controller. Detta kan kanske anses vara lite märkligt eftersom skaparna av Django uttryckligen hävdar att ramverket följer MVC-mönstret. Utvecklarna av Django anser att det är den underliggande koden i själva ramverket, vilken gör kopplingen mellan *requests* och vyer, som kan ses som *Controller*. Vidare hävdar de att benämningarna på de olika delarna i MVC-definitionen kan diskuteras samtidigt som det är den lösa kopplingen mellan delarna som är central och inte benämningarna.

View

Djangos definition på vad som är en vy skiljer sig också från den klassiska definitionen av MVC-mönstret. Själva benämningen vy förekommer men är bara en del av två vilka slutligen utgör det som presenteras för användaren. I Djangos värld är en vy något som bestämmer vilken data som ska presenteras för användaren däremot är dess uppgift inte att bestämma hur data ska presenteras rent visuellt, när det till exempel handlar om färg, typsnitt och så vidare. Istället används så kallade *Templates* vilka definierar, ofta med hjälp av *stylesheets* eller liknande, den exakta layouten för en viss data. Denna funktion gör det relativt enkelt att skapa en enhetlig layout om man utvecklar en större webbplats med ett stort antal sidor.

DRY

För att främja att utvecklare håller sig till DRY-filosofin använder sig Ruby on Rails av dels ”delmallar” (*partials* på engelska) och ”filter” (*filters*).

Partials används i Action View klasser för att förhindra att kod behöver upprepas. Om en applikation har flera vyer där en del av koden är likadan kan en programmerare skapa en ny fil till exempel: "_exempelmall.html.erb". Innehållet i denna kan sedan användas i flera vyer med hjälp av följande kommando:

```
<%= render :partial => "exempelmall" %>
```

För att förhindra koduppprepning i Action Controll klasserna förväntas man använda filter. Det finns filter som körs innan, efter eller före och efter en klass körs. Ett filter kan också ställas in att endast köras med vissa metoder angivna i klassen. Ett filter beskrivs lättast som ett metodanrop som körs på vissa tidpunkter beroende av typen av filter.

Ruby-on-Rails använder även av "layouts" för att förhindra koduppprepning. En "layout" används för att skapa grunden för utseende som sidan ska använda. Detta utseende kan sedan visas av vyer som använder sig av respektive layout. En layout är av typen ”.html.erb”. En

vanlig användning av layouts är att implementera HTML-kod för de delar av sidan som kommer se likadan ut i flera olika vyer samt att ange vilken *CSS stylesheet* som ska användas. Sedan indikerar man vart i layouten de olika vyerna ska renderas med följande tagg:

```
<%= yield %>
```

Djangos *Templates*-system är till stor del designat med *Don´t Repeat Yourself* i åtanke. I varje projekt definieras en lista med sökvägar till de platser som innehåller *Templates*. När en förfrågan tas emot går programmet igenom denna centrala lista och letar efter rätt *Template*. På så sätt undviks att *Templates* med samma funktionalitet finns utspridda på flera platser.

Templates kan också ärva från andra *Templates* för att, i så stor utsträckning som möjligt, undvika koduppprepning.

Applikationer är en annan del av Django som är tänkt att främja *Don´t Repeat Yourself* genom att de utformas på ett sätt som gör dem lätta att importera i andra projekt som insticksprogram. Här är det dock till stor del upp till den enskilde utvecklaren att genom de metoder som erbjuds åstadkomma denna lösa koppling.

Don´t Repeat Yourself återfinns också i betydligt mindre sammanhang som till exempel i URL-definitionerna.

Inbyggda funktioner

En av anledningarna till att använda ett ramverk för webbutveckling är att det oftast ger utvecklaren tillgång till en rad färdigbyggda funktioner som hanterar de vanligast förekommande funktionerna på en modern webbplats.

Utvecklarna av Django uttrycker det som att ramverket levereras med *Batteries included*[36]. Detta är ett engelskt uttryck som syftar på det faktum att de allra flesta funktioner som en utvecklare kan tänkas behöva använda finns inbyggda i ramverket i grundutförandet. Även programspråket Python som Django bygger på har också rykte om sig att innehålla en stor mängd inbyggda moduler för att hantera olika programmeringsuppgifter[37]. En stor del av denna funktionalitet är också tillgänglig i ett Django-projekt.

Även Ruby on Rails har ett stort *API* som inkluderar mycket av den funktionalitet som kan behövas när man utvecklar en webbapplikation. Det finns även tillgång till många insticksprogram på internet, som möjliggör att enkelt återanvända kod skapad av andra utvecklare och utöka funktionaliteten i Ruby on Rails.

En inbyggd funktion som är utmärkande för Django kallas för *Admin site* vilken kan aktiveras i varje projekt. Detta är en webbaserad funktion som genereras automatiskt där utvecklaren får tillgång till alla projektets modeller och enkelt kan lägga till eller ändra objekt. En stor fördel med detta är att innehåll som ska lagras i databasen kan börja läggas till i ett tidigt stadium av den som har tillgång till data samtidigt som utvecklaren färdigställer andra delar av webbplatsen så som layout. Denna funktion går att få i Ruby on Rails antingen genom att utveckla den själv eller använda några av de befintliga insticksprogram som finns tillgängliga. Någon inbyggd motsvarighet finns dock inte i Ruby on Rails.

Ruby on Rails har de båda JavaScript-ramverken *Prototype* och *Scriptaculous* inbyggda från start. Detta underlättar för utvecklare som behöver använda en stor andel JavaScript i sina projekt. Django har inget ramverk inbyggt men går att använda tillsammans med de flesta JavaScript-ramverk. Detta kräver dock en något större arbetsinsats samt kunskap från utvecklarens sida.

[36] http://docs.djangoproject.com/en/1.1/, inhämtat 2010-04-18

[37] http://www.python.org/about/, inhämtat 2010-04-18

Dokumentation och Support

Både Ruby on Rails och Django är relativt nya verktyg för att hantera dynamiskt innehåll inom webbutveckling om man jämför med andra äldre tekniker så som till exempel Java Server Pages. Detta gör att den mängd tryckt traditionell litteratur som finns tillgänglig är begränsad. En sökning på världens största nätbokhandel Amazon.com bekräftar detta. En sökning på Django genererar ett resultat på cirka 33 stycken böcker medan motsvarande siffra för Ruby on Rails är 116 stycken och för Java Server Pages 791 stycken.

För oss har dock inte bristen på tryckt litteratur inneburit något direkt problem eftersom vår främsta informationskälla, precis som för många andra, har varit Internet.

Ruby on Rails officiella hemsida finns på adressen *rubyonrails.org*. Där finns mycket officiell dokumentation för ramverket. Denna dokumentation består i huvudsak av en omfattande HTML-baserad API-referens som beskriver alla komponenter och dess metoder i en formell översikt. Varje metod har också en länk till den verkliga källkoden vilket kan vara användbart om det uppstår konstiga buggar när man använder en viss funktion.

Som komplement till denna formella referens finns det en avdelning som samlar olika typer av guider som genom steg-för-steg beskrivningar och exempel visar hur de olika funktionerna kan användas i olika situationer.

Förutom den officiella dokumentationen finns det också länkar till material skapat av användare. Dessutom har Ruby on Rails sin egen e-postlista, IRC-kanal samt en Wiki som samlar en mängd artiklar skrivna av användare.

Startpunkten för varje Djangoutvecklare är *The Django Projects* webbplats. Där finns dokumentation i form av manualer för de olika delarna, API-referens, steg-för-steg beskrivningar, kodexempel samt länkar till och information om andra resurser, både tryckta böcker samt andra hemsidor. Vidare finns det en gedigen lista på ett antal större sajter på Internet som utvecklats med hjälp av Django.

Utformningen av dokumentationen är till stor del exempelbaserad. Om man som utvecklare söker på Djangos hemsida efter en viss funktion länkas man oftast till en sida som beskriver hur den används i praktiken med hjälp av exempel. Detta bidrar också till att man snabbt kommer igång med att använda en viss funktion. Nackdelen är att det ibland kan vara svårt att få tillgång till all information eftersom det inte finns någon formell lista på alla funktioner och fält som tillhör en viss funktion.

Förutom dokumentationen finns det också hjälp att få via Djangos e-postlista. Antingen kan man söka bland frågor i arkivet eller så skickar man en ny fråga till gruppen. Djangos hemsida innehåller också en funktion som aggregerar blogginlägg som handlar om Django från hela Internet.

Skalbarhet

Skalbarhet är ett uttryck som kan förekomma i flera sammanhang när man talar om utveckling av webbplatser. Två av de viktigaste frågorna är dock hur underhållet påverkas när webbapplikationen och antalet filer växer, samt om ramverket är designat för att kunna hantera det stora antal förfrågningar som kan bli aktuellt när en större webbplats sätts i drift. I det senare fallet gäller det att det inte finns flaskhalsar i ramverket som hindrar det från att dra nytta av ytterligare tillförd hårdvara i form av servrar. En annan viktig del för att en applikation ska kunna skala väl är att utvecklare har möjlighet att skriva effektiv testkod för att säkerställa önskat resultat av de komponenter som utvecklas.

Ruby on Rails skiljer sig inte nämnvärt från andra populära ramverk när det handlar om inbyggda funktioner för att hantera då applikationen växer och antalet förfrågningar ökar[38]. Framförallt brukar man prata om horisontell skalning där man använder en så kallad *load-balancer* som kan fördela ett stort antal förfrågningar mellan ett antal bakomliggande duplicerade webbservrar. Detta gör att man kan dra nytta av tjänster som RightScale vilka gör det enkelt att utöka kapaciteten vid tillfälliga toppar i trafiken[39].

Den vanligaste metoden i dagsläget som används för att publicera en Ruby on Rails applikation är att använda en modul för Apache webbserver. Det har dock varit svårt att hitta information om exakt hur denna teknik hanterar skalningsproblematiken.

När det gäller Ruby on Rails så finns det flera stora sajter som är utvecklade med det som grund. Några av de kanske mest kända är mikrobloggen Twitters[40] hemsida samt den amerikanska videotjänsten Hulu[41]. Detta tyder på att Ruby on Rails är möjligt att använda för större projekt.

Ruby on Rails har testning som en integrerad del och använder unit test. När man använder kommandot scaffolding för att skapa nya komponenter i Ruby on Rails så skapas även kod för att möjliggöra testing av dessa komponenter. Det finns även stöd för att integrera testningen med databaser. Användaren lägger då till testdata i en databas som sedan verifieras med de berörda komponenterna.

Django är, enligt den officiella dokumentationen, designat med skalbarhet i åtanke. Ramverket använder en arkitektur kallad *shared-nothing* vilket kort innebär, när man pratar om distribuerade datasystem, att varje del eller nod är oberoende och självförsörjande[42]. Detta medför att det är möjligt att tillföra extra hårdvara på olika nivåer, oberoende, så som databasservrar eller webbservrar.

Django har stöd för automatisk testning genom antingen Doctest eller Unit test, vilka är samma testsystem som återfinns i programmeringsspråket Pythons standardbibliotek. Automatisk testning är idag en vital del i varje större utvecklingsprojekt för att kunna säkerställa en viss tillförlitlighet i applikationen[43]. För många webbaserade tjänster, så som banktjänster, är detta ännu viktigare.

På *djangosites.org* finns en lista över webbplatser som använder Django. Denna lista innehåller allt från mindre hobbyprojekt utvecklade av en ensam person till stora nyhetssajter i USA med flera tusen besökare dagligen. Detta ger en indikation på att Django har ett brett användningsområde.

Databas

Django och Ruby on Rails har stöd för databaserna MySQL, PostgreSQL samt SQLite som standard. Django har även inbyggt stöd för databasen Oracle. Databaserna MySQL, PostgreSQL och Oracle i listan är fullfjädrade applikationer som klarar de flesta tillämpningar medan SQLite är en enklare variant som lämpar sig för utveckling och i mindre projekt. Förutom dessa finns det även ett antal tredjepartsmoduler som gör det möjligt att använda en handfull andra populära databaser.

---

[38] http://www.buildingwebapps.com/articles/6419-can-rails-scale-absolutely, inhämtat 2010-04-07

[39] http://www.rightscale.com/info_center/, inhämtat 2010-04-18

[40] http://www.twitter.com/

[41] http://www.hulu.com/

[42] Michael Stonebraker, The Case for Shared Nothing, University of California, 1986

[43] William G. J. Halfon, Web application modeling for testing and analysis, Foundations of Software Engineering, ACM, 2008

Som nämnts tidigare erbjuder både Ruby on Rails och Django ett API för kommunikation med databaser. Detta API förenklar interaktionen med databasen genom att all sådan programkod skrivs med vanlig Python- respektive Rubykod. Om det skulle behövas så finns det även möjlighet att inbädda vanliga SQL frågor i koden. Nackdelen med denna metod är dock att det kan innebära problem om man i framtiden skulle vilja byta databas. Applikation är då inte längre databasoberoende. Används det inbyggda gränssnitten erhålls istället en väldigt lös koppling till databasen och applikationerna blir plattformsoberoende till databasen.

Diskussion och slutsats

Under vår jämförelse av Ruby on Rails och Django har vi belyst vissa resultat. I detta avsnitt har vi valt att diskutera dessa resultat mer ingående samt att ge vår slutsats kring valet av ramverk.

Vår jämförelse har visat att det finns stora likheter mellan Django och Ruby on Rails. Detta är kanske inte så konstigt eftersom de båda tillhör samma grupp av ramverk som på engelska brukar kallas *agile web development frameworks*. Med detta syftar man på det faktum att de är inriktade på att snabba upp utvecklingsprocessen genom att erbjuda färdiga byggblock samtidigt som de förespråkar en god design av koden.

	Ruby on Rails	**Django**
Programmeringsspråk	Ruby	Python
Databasstöd	MySQL, PostgreSQL, SQLite, IBM DB2, Microsoft SQL Server, Oracle, Sybase, Firebird	MySQL, PostgreSQL, SQLite, Oracle
Stöd för migrering och evolution av databasscheman	Inbyggt ramverk för att hantera detta.	Begränsat stöd.
Templatesspråk	Inbäddad Ruby-kod	*Django Template Language*
Stöd för automatiserad testning	*Unit tests*, är automatiskt inbyggt i varje applikation.	*Doctests, Unit tests*
Automatiskt genererad *Admin*-funktion	Nej	Ja
Officiell dokumentation	Detaljerad men inte så många exempel.	Exempelbaserad
Stöd för insticksprogram (tredjepart)	Väl utvecklad community.	Det finns, men inte lika välutvecklat.
Stöd för Javascript-ramverk	Inbyggt JavaScript-ramverk, *Prototype, Scriptaculous*.	Inget inbyggt ramverk, dock stöd för de flesta.
Fri källkod	Ja	Ja

Tabell 4: Sammanfattning av resultat från jämförelsen mellan Ruby on Rails och Django.

En aspekt som kan vara viktig att beakta är de båda ramverkens ursprung. Django är som nämnts tidigare utvecklat för att underlätta driften av flera större nyhetssajter. Det vill säga webbplatser där driften till stor del handlar om att publicera och underhålla en stor mängd innehåll i olika former så som artiklar, bilder och liknande. Ruby on Rails, å den andra sidan,

uppstod i samband med utvecklingen av webbapplikationen BaseCamp som är en applikation för projektsamarbete. BaseCamp påminner mer om en vanlig skrivbordsapplikation än en webbsida och förlitar sig på en hel del interaktivitet. För att kunna erbjuda denna interaktivitet används i huvudsak JavaScript vilket Ruby on Rails har inbyggt stöd för. Django kan också använda sig av JavaScript men här krävs ett större mått av manuellt arbete från utvecklarens sida. Detta kan vara en betydande faktor i valet av ramverk.

Om valet står mellan dessa två ramverk så är det kanske inte en viss funktion som återfinns i det ena och inte i det andra ramverket som avgör utan kanske snarare tidigare erfarenhet. Eftersom syntaxen för de under liggande programmeringsspråken, Python samt Ruby, skiljer sig åt i stor utsträckning bör man som utvecklare fundera på om det inte är bättre att välja ett ramverk baserad på ett språk som man har erfarenhet av.

Även när det gäller inbyggd funktionalitet finns det inga större skillnader utan båda ramverken innehåller det mesta som en utvecklare kan tänkas behöva. En betydande funktion som återfinns hos Django men inte hos Ruby on Rails är den speciella automatiska *admin-funktionen* vilken sparar in en hel del utvecklingsarbete.

De båda ramverken skiljer sig ganska väsentligt i hur de hanterar den problematik som uppstår när data som modelleras i databasen förändras. Detta brukar benämnas som ”evolution av databasscheman”. Ruby on Rails har här en fördel eftersom det använder en teknik för att utifrån själva databastabellerna identifiera de attribut som hör till en viss modell. I Django måste utvecklaren explicit specificera attributen i samband med att en modellklass definieras. Ruby on Rails erbjuder även ett inbyggt ramverk med färdiga funktioner för att hantera migrering och evolution av databasen. I Django är detta stöd minimalt. Ruby on Rails får därmed anse bättre möte de krav på flexibilitet utvecklare ofta stöter på i sina projekt där specifikationer ändras kontinuerligt.

Dokumentationen är en punkt där de två ramverken skiljer sig något. Här upplevdes det som att Djangos exempelbaserade dokumentation, vilken inspirerade till användning av de olika funktionerna, stämde bättre överens med filosofin om snabb och effektiv utveckling. Här kändes Ruby on Rails något gråa och formella, om än effektiva, dokumentation som mindre inspirerande. Trots dessa anmärkningar råder det ändå knappast någon brist på dokumentation och inspiration i alla fall om man använder Internet som sin huvudsakliga källa.

Vi anser att det är svårt att ge några generella råd till utvecklare angående vilket ramverk man ska välja till sitt nästa projekt. Valet av ramverk är i allra högsta grad beroende av tidigare erfarenheter samt projektets krav och omfattning. Vi har därför valt att lyfta fram några av de attribut hos ramverken som vi tror kan ha en påverkan i valet av ramverk.

Referenslista

Böcker

DANIEL CAZZULINO, VICTOR GARCIA APREA, JAMES GREENWOOD AND CHRIS HART, *Beginning Visual Web Programming in VB .NET*, Apress, 2005

MARK PILGRIM, *Dive into Python 3*, Apress, 2009

SAN MURUGESAN, *Web Application Development: Challenges and the Role of Web Engineering,* utgiven i *Web Engineering: Modelling and Implementing Web Applications*, Springer, 2008

Artiklar

ALFRED WAI-SING LOO, *Peer-to-Peer Computing*, Springer London, 2007

JOSÉ IGNACIO FERNÁNDEZ-VILLAMOR; LAURA DÍAZ-CASILLAS; CARLOS Á. IGLESIAS, *A comparison model for agile web frameworks*, Univ. Politécnica de Madrid, ACM, 2008

LEESA MURRAY; DAVID CARRINGTON; PAUL STROOPER, *An approach to specifying software frameworks, The University of Queensland, 2004*

MEHDI JAZAYERI, *Some Trends in Web Application Development*, IEEE Computer Society, 2007

MICHAEL STONEBRAKER, *The Case for Shared Nothing*, University of California, 1986

ROY THOMAS FIELDING, *Architectural Styles and the Design of Network-based Software Architectures*, http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, inhämtat 2010-03-24

STEFAN KOCH, *Extreme Programming and Agile Processes in Software Engineering*, Springer Berlin/Heidelberg, 2004

WILLIAM G. J. HALFON, *Web application modeling for testing and analysis*, Foundations of Software Engineering, ACM, 2008

Webbmaterial

37SIGNALS, http://basecamphq.com/, inhämtat 2010-05-01

ADRIAN HOLOVATY AND JACOB KAPLAN-MOSS, http://djangobook.com/en/2.0/chapter12/, inhämtat 2010-04-12

APACHE HTTP SERVER PROJECT, http://httpd.apache.org/, inhämtat 2010-05-02

BRUCE STEWART, http://linuxdevcenter.com/pub/a/linux/2001/11/29/ruby.html, inhämtat 2010-03-21

CUNNINGHAM & CUNNINGHAM, INC., http://c2.com/cgi/wiki?DontRepeatYourself, inhämtat 2010-04-14

DJANGO SOFTWARE FOUNDATION, http://docs.djangoproject.com/en/1.1/faq/general/, inhämtat 2010-04-18

DJANGO SOFTWARE FOUNDATION, http://docs.djangoproject.com/en/1.1/faq/general/, Inhämtat 2010-04-18

DJANGO SOFTWARE FOUNDATION, http://docs.djangoproject.com/en/1.1/, inhämtat 2010-04-18

DJANGO SOFTWARE FOUNDATION, http://docs.djangoproject.com/en/1.1/intro/tutorial01/#intro-tutorial01, inhämtat 2010-03-27

TIM BRAY, *Comparing Frameworks,* http://www.tbray.org/ongoing/When/200x/2006/11/10/Comparing-Frameworks, inhämtat 2010-04-18

JOSH KENZER, http://www.radicalbehavior.com/5-question-interview-with-twitter-developer-alex-payne/, inhämtat 2010-04-28

MICHAEL BÄCHLE; PAUL KIRCHBERG, *Ruby on Rails,* IEEE Software, 2007

MICHAEL SLATER, http://www.buildingwebapps.com/articles/6419-can-rails-scale-absolutely, inhämtat 2010-04-07

ORACLE, http://java.sun.com/blueprints/patterns/MVC-detailed.html, inhämtat 2010-04-28

PHUSION, http://www.modrails.com/, inhämtat 2010-04-28

PHUSION, http://www.rubyenterpriseedition.com/, inhämtat 2010-04-28

RIGHTSCALE, INC., http://www.rightscale.com/info_center/, inhämtat 2010-04-18

RUBY ON RAILS, http://guides.rubyonrails.org/, inhämtat 2010-04-20

RUBY ON RAILS, wiki.rubyonrails.org, inhämtat 2010-04-20

RUBY ON RAILS, http://guides.rubyonrails.org/getting_started.html#getting-up-and-running-quickly-with-scaffolding, inhämtat 2010-04-20

RUBY VISUAL IDENTITY TEAM, http://www.ruby-lang.org/, inhämtat 2010-05-01

Officiella hemsidan för programmeringsspråket Ruby.

STEFAN TILKOV, http://www.infoq.com/articles/rest-introduction, inhämtat 2010-04-08

THE PYTHON SOFTWARE FOUNDATION, http://www.python.org/doc/faq/general/, inhämtat 2009-04-15

THE PYTHON SOFTWARE FOUNDATION, http://www.python.org/about/, inhämtat 2010-04-20

TWITTER, http://www.twitter.com, inhämtat 2010-05-01

TWITTER, http://engineering.twitter.com/2010/03/unicorn-power.html, inhämtat 2010-04-28