Analys av programspråket Go

CAJ HOFBERG
och JOEL SMEDBERG

KTH Datavetenskap
och kommunikation

Examensarbete
Stockholm, Sverige 2011

Analys av programspråket Go

C A J H O F B E R G
o c h J O E L S M E D B E R G

Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik
Kungliga Tekniska Högskolan år 2011
Handledare på CSC var Alexander Baltatzis
Examinator var Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2011/
hofberg_caj_OCH_smedberg_joel_K11083.pdf

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

Go is a programming language developed by Google. The language is new and aims to replace the more established languages. Parallel programming, computer networking and componentbased programming has been improved, according to the developers. The language's quality, agility, suitability for small projects and future prospects are treated in this paper. Go is compared to other languages in terms of self-documentation, productivity and future prospects. The authors, with no previous experience of Go, has acted as test subjects and implemented a wiki in Go. The time spent has been documented and the programming has been analyzed from the four perspectives of a SWOT analysis. A quantitative questionnaire to evaluate the view of Go has been conducted. Go did not appear to be the revolutionary new language we expected.

Referat

Go är ett programmeringsspråk utvecklat av Google. Språket är nytt och ämnar ersätta etablerade språk. Parallellprogrammering, nätverkskommunikation och komponentbaserad programmering har bäddats in. Språkets kvalité, smidighet, lämplighet för små projekt och framtidsutsikter behandlas i uppsatsen. Go jämförs mot andra språk i fråga om självdokumentation, produktivitet och framtidsutsikter. Författarna, utan tidigare erfarenheter av Go, har agerat som försökspersoner, "försöksprogrammerare", och implementerat en wiki i Go. Nedlagd tid har dokumenterats och utvecklandet analyserats ur de fyra perspektiven i en SWOT-analys. En kvantitativ enkät med kryssfrågor som utvärderat synpunkter om Go har genomförts. Go gav inte intryck av att vara det banbrytande språk vi hade hoppats på.

Statement of collaboration

Sammanfattningarna på svenska och engelska, definitioner, teorikapitlet och utformningen av SWOT-tabellen skrevs av Hofberg, C. Enkäten med tillhörande diagram, större delen av analysen och diskussionskapitlet utformades i huvudsak av Smedberg, J. Inledningen, slutsatsen och metodkapitlet skrevs gemensamt av båda författarna. Båda författarna har agerat som växelvis observatörer och försökspersoner i SWOT-analysen. Observatören mätte tiden och matade in information i tabellen och försökspersonen installerade, körde, programmerade och studerade. Flera uppgifter har gjorts om; i de fallen har de mest rättvissande resultaten tagits med.

Innehåll

Kapitel 1

Inledning

1.1 Definitioner

Go - Med Go avses programmeringsspråket Go, utvecklat av Google

Komponentbaserad programmering - En form av mjukvaruutveckling som delar upp programkod i komponenter[1]

IDE - Integrerad utvecklingsmiljö; en kombination av textredigerare, kompilator och debugger[2]

1.2 Sammanhang

Go är ett programmeringsspråk som ämnar hjälpa programmerarna att bli mer produktiva. Meningen är att förenkla för parallellprogrammering och kommunikation i nätverk. Modulär programkonstruktion uppmuntras. Syftet med språket är att förbättra populära programspråk med ändringar som har varit svåra att implementera i andra språk. En ren stil är ett centralt mål för utvecklarna. Ett problem som löstes med kompromisser var att det var svårt att förbättra gamla språk som till exempel C++ eller ändra layouten för koden. Man har bestämt sig från språkutvecklarsidan att skapa ett nytt språk som ämnas ha ett annorlunda upplägg fast som ändå går att känna igen sig i för erfarna programmerare som är vana vid det äldre upplägget.[3] Flera trender gav upphov till idén att designa ett nytt språk från grunden. Man la märke till att datorer i ökad omfattning har börjat levereras med stöd för samtidig körning i en processor. Trots att datorerna har blivit mycket snabbare har inte utvecklingsarbetet för kod blivit snabbare. Garbage collection och samtidig körning lyfts också fram som viktiga faktorer för ett framtida språk. Betydelsen av hierar-

[1] http://en.wikipedia.org/wiki/Component-based_software_engineering
[2] http://en.wikipedia.org/wiki/Integrated_development_environment
[3] http://en.wikipedia.org/wiki/Go_%28programming_language%29

kier har minskat även om man vill göra objektorienterad programmering möjlig. Typerna i Go är mindre strikta än i mer använda språk.[4]

1.3 Bakgrund

Många rykten om Go finns men det är i nuläget svårt få en god uppfattning om språkets kvalité utan att testa språket personligen. Denna undersökning syftar till att räta ut några av de frågetecken som finns samt att ge en objektiv bild av språket som helhet. Bara att visa att språket är lätt att utveckla i för en erfaren Go-programmerare behöver inte vara det enda intressanta att studera. Det skulle kunna visa sig att språket är tidsmässigt effektivt att arbeta med men svårt att lära sig att använda eller svårt att programmera i även för den som är van. Information saknas om hur lång tid olika typer av uppgifter tar att utföra med hjälp av programspråket inom de projekt som involverar mjukvaruutveckling som på något vis är beroende av eller underlättas av att utnyttja det aktuella språket som verktyg. Möjligen skulle vissa typer av sysslor gå snabbt med språket och andra mindre snabbt. Med hjälp av en kartläggning av skillnader i tidsåtgång skulle planeringen underlättas liksom valet av programspråk till olika utvecklingsprojekt. Dessutom kan det finnas ett intresse av att utvärdera hur väl språket fungerar för praktiska sammanhang såsom utveckling av enkla servrar. Även om Go endast skulle visa sig vara väl anpassat för specialiserade forskningssammanhang är det värdefull information för de institutioner som kan tänkas behöva utveckla annorlunda mjukvara. I så fall behövs kartläggning av vad språket främst kan ha för framtida användning inom sådana miljöer.

För att veta om språket i realiteten är värt att lära sig krävs även en analys av dess framtidsutsikter. Även om Go har många tekniska finesser avgör detta tyvärr en förhållandevis liten del av språkets framgång. Andra avgörande faktorer inkluderar inte minst fortsatt utveckling av språket, budget för marknadsföring, hur många större projekt som byggs i det, inlärningströskel, och om språket lyckas skapa ett gott rykte kring sig. Även om befintliga programmeringsspråk utvecklats en hel del under de senaste tio åren har inget större nytt språk trädit fram. Go har även kopplingar till Google, som är ett företag med stora möjligheter att spendera resurser på diverse projekt i reklamsyfte. För en värderare alternativt marknadsförare vid Google skulle en noggrann utomståendes utvärdering av språkets framtid vara väsentlig. Om det skulle visa sig att det i stort är lönsamt att utveckla eller stödja programspråk för marknadsföringsföretag eller sökmotorföretag skulle det kunna förändra bilden av hur resurserna för marknadskommunikation ska fördelas. Idag finns redan större mjukvaruutvecklingsföretag såsom Microsoft som satsar på utvecklingsspråk och tillhörande miljöer, till exempel .NET[5], men då säljs programmen för utveckling av programmen och mjukvaran som behövs för att kunna köra programmen,

[4] http://golang.org/doc/go_faq.html
[5] http://sv.wikipedia.org/wiki/Dotnet_Framework

till exempel Windows, av samma företag. I dagsläget annonseras ingen efterfrågan på jobb för programmerare i Go på någon av de större svenska jobbsidorna. Dock anger Golang.org, kompilatorns officiella hemsida, att Google använder Go för ett flertal interna projekt.[6]

1.4 Problemformulering

Hur står sig Go gentemot andra programmeringsspråk vad gäller självdokumentation, produktivitet och framtidsutsikter?

1.5 Mål

Vissa språk går att förstå utan vana. Vissa programspråk kan behöva extra mycket vägledning eller separat dokumentation för att vara av värde. Hur väl kod skriven i Go går att förstå utan ytterligare dokument eller information ska därför tas reda på. Målet är dels att undersöka hur lång tid utvecklingen tar och dels att ta reda på hur snabbt typiska program kör. Tiden det tar för en utvecklare att skriva webbapplikationer i Go önskas få fram. Att utvärdera Go som kommande programmeringsspråk, förstå dess grunder samt jämföra dess styrkor respektive svagheter mot andra programmeringsspråk vore mycket värdefullt för att avgöra om språket är mödan värt att behandla.

1.6 Avgränsningar

Avancerade funktioner i Go, annat än vad som behövs för att programmera en wiki, använda flera trådar eller andra för en utvärdering kritiska funktioner kommer inte analyseras. De för utvärdering skrivna programmen kommer hållas små och antalet utvecklare per projekt kommer inte att överstiga två. En bättre uppfattning om språkets skalbarhet till större projekt kommer utebli, utöver de delar som krävs för att testa komponentbaserad programmering. Resultatet av denna studie kommer i första hand vara en fingervisning om språkets tekniska potential att lyckas. Någon analys av politik, ekonomisk satsning eller dylika faktorer som med all sannolikhet spelar stor roll kommer inte alls behandlas.

[6] http://golang.org

Kapitel 2

Teori

2.1 Reliabilitet/Validitet

Thurén talar om reliabilitet och validitet. Reliabilitet innebär att siffror som tagits fram är korrekta och inte är missvisande. Till exempel om försöksgruppen är för liten blir reliabiliteten för låg.[7] Det är viktigt att enkäter i undersökningar utformas så att de har en hög nivå av reliabilitet. Många personer som är slumpmässigt utvalda och representerar rätt målgrupp är att önska. Att få hög validitet kan vara svårare. Med validitet menas att de framtagna måtten är relevanta för undersökningen.[8] För att få högre reliabilitet bör man inte låta yttre faktorer störa i ett formulär såsom att före utlämningen av formulären berätta om vad man anser om språket. För att få högre validitet bör man fråga om det aktuella programspråket (i vårt fall Go) och inte fråga om programmering i allmänhet. SWOT-analysen bör inte behandla områden utanför de fyra områdena som ska ingå i en standardiserad SWOT-analys. Annars riskerar validiteten att bli låg. För att erhålla en hög reliabilitet i SWOT-analysen bör försöksprogrammeraren intervjuas direkt efter programmeringen så att försöksprogrammeraren inte hinner glömma bort viktiga detaljer. Enkäten och analysen har skett i enlighet med ovanstående.

2.2 Induktion/deduktion

Vi använder oss explicit av induktion eftersom information endast insamlas utan möjlighet att säkerställa att all information är korrekt och utan att informationen utesluter alla andra alternativ.

2.3 Positivism/hermeneutik

Det finns två huvudinriktningar inom vetenskapen. Positivismen är traditionellt av naturvetenskaplig härkomst. Hermeneutiken är traditionellt frambringad av huma-

[7] Thurén, Vetenskapsteori för nybörjare, 2 upplagan, 2007, Liber, Malmö, s. 34
[8] ibid s. 26

nister. Positivismen går mer ut på att använda säker kunskap och uppmuntrar till att kvantifiera fakta. Kvantifiering innebär att fakta ska analyseras statistiskt och göras om till värden som går att mäta. Målet är att få en säker kunskap.[9] Ofta används mätvärden av positivister. Hermeneutiken går däremot ut på att förstå varför skeenden är som de är. Målet är ofta att förklara människor och varför de beter sig som de gör. Det är svårt att avgöra om hermeneutikern har rätt eller fel i sina resonemang. Det blir viktigt att det man tolkar sätt in i rätt kontext (sammanhang).[10] Djupintervjuer är vanligare i hermeneutisk forskning. Enkäterna har studerats positivistiskt. Kryss räknades varefter antalet kryss analyserades snarare än att utseendet på svaren analyserades. SWOT-analysen var hermeneutiskt upplagd. Försöksprogrammeraren intervjuades och ombads ge kritik på utvecklandet ur olika perspektiv.

2.4 Empiri

Empiri är vanligt inom positivismen och går ut på att istället för att granska tidigare kunskap skapa en egen uppfattning från grunden utifrån verkligheten.[11] Samtliga genomförda undersökningar i detta dokument är empiriska då ingen nämnvärd tidigare kunskap finns att tillgå.

2.5 Kvalitativ utvärdering

Då värdesättning av kod riskerar att bli svår att kvantifiera i rena siffror, behövs en kvalitativt orienterad undersökning av de bitar som inte kan kvantifieras. Därför valdes en standardiserad analysmodell för den kvalitativa analysen. Den standardiserade analysmodell vi använde var SWOT-modellen. I och med att enkäten kan utformas så att den har kryssfrågor som svarsalternativ kan kryssen räknas. På så vis blev enkätsvaren kvantifierbara och utvärderingen av enkäten var kvantitativ.

[9] Thurén, Vetenskapsteori för nybörjare, 2 upplagan, 2007, Liber, Malmö, s. 94-103
[10] ibid s. 34
[11] ibid s. 18-21

Kapitel 3

Metod

För att testa enkel programmering och förmågan att utveckla enkla program som körs flertrådat samtnätverkskommunikation har vi byggt en enkel wiki i Go. För att lyckas med ovanstående krävs en grundläggande förståelse för Go som programmeringsspråk. En noggrann bokföring över nedlagd tid under utvecklingen, även tid lagd på manualer och guider, ger en uppfattning om hur svårt språket är att lära sig i praktiken. Våra kunskaper i Go var obefintliga vid projektets inledning, så vid tog tillfället i akt och bokförde vår egen lärandeprocess. I studien ska typning, kompilering, parallellkörning, säkerhet och utvecklingsverktyg utvärderas och tid bokföras separat. För att utvärdera folks tilltro till språket ska vi i enkäterna utvärdera hur stor potential språket kan ha och om de anser att det verkar vara ett bra språk.

Thurén beskriver att värderingar kan påverka slutsatser. Det finns så kallade värdepremisser som måste isoleras och utvärderas.[12] Eftersom våra enkäter bedöms behandla enkätsvararnas värderingar av språket kommer en sådan premiss förekomma. Utöver det kan undersökningen av språket leda till att undersökaren får värderingar om språket. Därför måste även sådana värderingar hänföras till värdepremisser som sedan kan sorteras ut från ren fakta.

Formulären ska ha en positivistisk karaktär och analysen av programspråket under wikiutvecklingen en mer hermeneutisk. I och med att positivismen inte är ämnad för att analysera människors handlingar bör allt utanför svarsformulären som inte följer mallen beaktas med låg prioritet. Wiki-utvecklingen ska analyseras på ett sätt som inte känns luddigt men som ändå kan ta hänsyn till mänskliga faktorer. Hur mycket tid som läggs ner är en kvantifierad fakta men analysen av den faktan kan vara hermeneutisk trots faktans natur i och med att kännedom finns om vad siffrorna representerar i verkligheten. Med andra ord: Ingen djupanalys av text skriven utanför kryssrutor och liknande svarselement har genomförts. Inte bara studier i samband mellan siffror har ingått. Programmerandet har analyserats i detalj.

[12] Thurén, Vetenskapsteori för nybörjare, 2 upplagan, 2007, Liber, Malmö, s. 52

3.1 Materiel

3.1.1 Teknisk utrustning

Stöd för Windows saknas i nuläget för Go-kompilatorn. Datormiljöer som stödjer Linux eller Mac OS X behövdes. Hårdvarumässigt behövdes en dator med flera kärnor för att kunna dra nytta av den effektiva trådhantering som utlovats. I dagsläget är mjukvaran körbar på vanliga, standardutrustade datorer med Linux då kraven på miljön mest rör vanligt förekommande program.

3.1.2 Intervjupersoner

I skrivande stund ges inte ens några relevanta träffar vid sökning på programmering i Go i Sverige. Någon aktiv svensk utveckling ser inte ut att pågå för tillfället. Vi gjorde därför bedömningen att en erfaren Go-programmerare visade sig för svår att få tag på. Istället kommer enkäten att omfatta de i enkäten intervjuades bild av framtidsutsikterna för språket.

3.2 Kartläggning av tidsåtgång

Eftersom forskningsgruppen består av två medlemmar har en person ansvarat för tidtagning och en annan utfört uppgifterna. Samtliga tider är angivna i tabellen för varje uppgift.

3.3 Modell för analys av fördelar/nackdelar

Normalt används modellen för utvärdering av marknadsföring och inom många olika typer av projekt. Strengths (interna fördelar) tolkar vi som fördelar relaterade till teknik och Opportunities (externa fördelar) som fördelar relaterade till att attrahera och behålla programmerare. Motsvarande gäller för Weaknesses (interna nackdelar) som vi tolkar som nackdelar relaterade till teknik och Threats (externa nackdelar) som nackdelar relaterade till att attrahera och behålla programmerare. Tabellen används för att samla in alla åsikter och tankar som uppkom i och med kodningen. Den fyller också funktionen att det är lättare att täcka alla områden och se arbetet ur flera olika synvinklar samtidigt som utvecklaren uppmuntras ta på sig en växelvis positiv och negativ roll under kritiken. Utöver det används det interna perspektivet för att studera tekniska kvaliteter och det externa för att analysera för- och nackdelar i tillämpningar.

3.4 Kartläggning av fördelar/nackdelar

SWOT-modellen valdes för att analysera hur mjukvaruutvecklingen fortskred. För utvärdering av språkets olika för- och nackdelar har för denna bit valts en mer kvalitativ, ingående analys av språkets respektive delar.

3.5 Utformning av enkäten

På samtliga frågor ges svarsalternativet "annat"; detta ger enkätsvararen en chans ge ett eget svar om inget av de listade passar. Avsaknad av en sådan fråga riskerar att tvinga fram ett svar från användaren som denne annars inte hade givit. Utformningen av frågorna bygger i grund och botten på skaparnas antagande av vad folk kommer att svara. Ett valfritt alternativ syftar därmed till att fånga in eventuella misstag vid utformningen av enkäten.

Skulle undersökningen utökas till att omfatta även andra studieinriktningar och andra årskurser, är det inte otänkbart att de olika grupperna överlag skulle svara olika på frågorna. Att fånga upp sådana trender bland respektive program och årskurs skulle ge en mer rättvisande bild. En djupare analys av hur samt varför olika grupper tror olika ligger utanför denna undersökning; vi nöjer oss med att ta höjd för att fenomenet kan förekomma. Att finna en lämplig målgrupp för enkätundersökningen är långt ifrån trivialt. Det skulle till exempel kunna vara sannolikt att om alla tillfrågade hade iPhone-programmering som sitt främsta intresse, skulle intresset för Go skilja sig markant från det av programmerare i allmänhet. För att få en mer nyanserad uppfattning om läget för programspråket valde vi en dataklass, en grupp kan antas vara intresserade av programspråk. Av dataklasserna valdes första årskursen eftersom många i den kategorin ännu inte har specialiserat sig på specifika språk eller inriktningar. För få svarsalternativ skulle ha riskerat att bli ett väldigt trubbigt sätt att undersöka på, samtidigt som alltför många alternativ tenderar att få undersökta personer att tröttna på undersökning.[13] En balans däremellan är således på sin plats, efter övervägande valdes fem kryssfrågor per fråga.

För att få ett mer objektivt urval av vilka programmeringsspråk att ta med för jämförelse användes Tiobe.com:s[†] index över mest använda programmeringsspråk. Ordningen på listan har sedan slumpats innan den placerades i enkäten. För enkätundersökningen valdes att inte använda några kodexempel. Detta hade kanske kunnat vara intressant för att mäta hur intuitivt ett språk är jämfört med ett annat, men hade medfört en del komplikationer. Förståelse för kodsnuttar beror helt på hur mycket tid man lägger ned på att sätta sig in dem, samt tidigare erfarenhet av liknande språk. Skulle en sådan undersökning göras väl, krävs att deltagarna i undersökningen verkligen lägger ned ungefär lika mycket tid på varje fråga för att det överhuvudtaget skall gå att jämföra. Ett annat problem är att vissa kodsnuttar blir klart mer lättlästa i vissa än i andra språk, mycket beroende på hur kompakt programmeraren valt att uttrycka sig. "i = i + 1" är till exempel mer intuitivt än "i++". En jämförelse mellan två kodstycken fordrar alltså att koderna är ungefär på samma nivå. Programmeringsspråket Java ingick inte i enkäten eftersom enkäten ämnades delas ut under en Java-föreläsning.

[13] Dahmström, Från datainsamling till rapport, 3 upplagan, 2000, Studentlitteratur AB, Lund

[†] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

3.5.1 Kvantitativ utvärdering

För utformningen av enkätundersökningen har det valts ett fokus på kvantitativ analys. Frågorna har försökts att utformas på sådant sätt att de relativt enkelt kan översättas till ett siffervärde, där andra människors svar summeras och jämförs. Enkäten är primärt uppbyggd av kryssfrågor; detta för att minimera den tid som krävs för att fylla i enkäten. Fritextsfrågor tenderar att bli överhoppade medan ett kryss i en ruta går betydligt snabbare och lägger mindre krav på att personen i fråga skall tänka ut någon bra formulering. Frågor där man ombeds ordna en lista från bäst till sämst blir problematisk då personen ofta har en bestämd åsikt om vad som är bäst men inte direkt kan uttala sig om övriga alternativ. Den nedre delen av listan hamnar då i mer eller mindre godtycklig ordning eller uteblir helt.[14] Det blir då svårt att jämföra en enkät med andra enkäter.

[14]Dahmström, Från datainsamling till rapport, 3 upplagan, 2000, Studentlitteratur AB, Lund

Kapitel 4

Resultat

4.1 Tabell för tidsåtgång

Observatören mätte tiden och matade in information i tabellen och försökspersonen installerade, körde, programmerade och studerade.

Pass	Tidsåtgång	Starttid	Sluttid	Försöksperson
Fixa kompilator	20 min	18.10	18.30	C. Hofberg
Kompilera HelloWorld	3 min	14.07	14.10	J.Smedberg
Wiki: Datastrukturer & Filhantering	27 min	14.12	14.39	J.Smedberg
Wiki: HTTP-server	10 min	14.50	15.00	J.Smedberg
Wiki: Integrera (3) och (4)	41 min	15.15	15.56	J.Smedberg
Wiki: Kunna ändra & spara sidor	33 min(2 st)	15.31	16.04	J.Smedberg
Wiki: Mallar i Go	33 min(2 st)	15.31	16.04	J.Smedberg
Wiki: Felhantering	9 min	16.04	16.13	J.Smedberg
Wiki: Cachning	12 min	16.15	16.22	J.Smedberg
Wiki: Säkerhetsluckor	4 min	19.39	19.43	J.Smedberg
Wiki: Återanvändning av kod	10 min	19.43	19.53	J.Smedberg

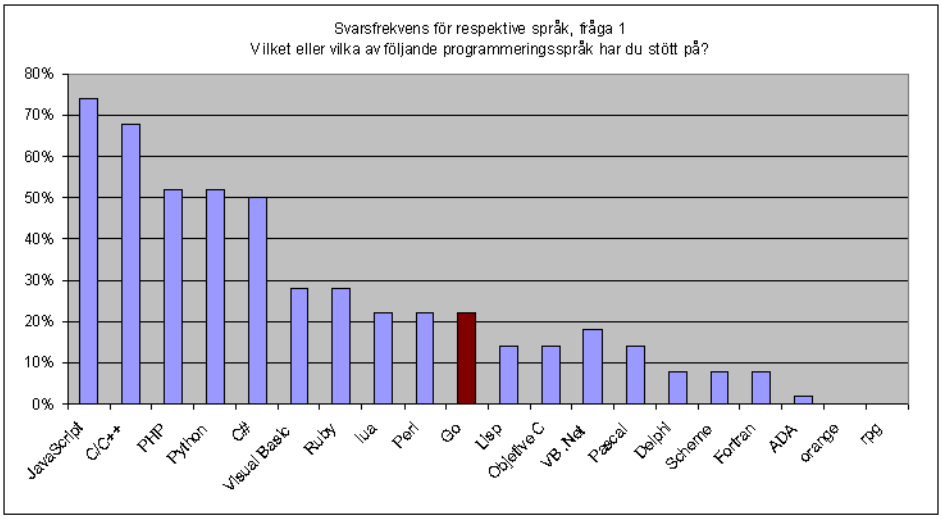
4.2 SWOT-tabell

Observatören la märke till kommentarer som försökspersonen yttrade under försöken och sammanställde sedan information. Luckor som var tomma försökte fyllas i genom intervjuer parterna emellan.

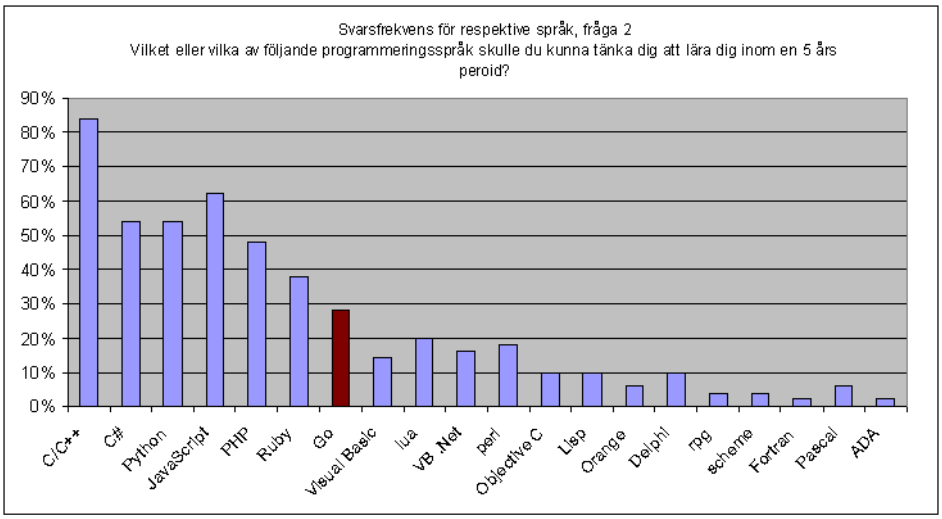
Pass	Strengths	Weaknesses	Opportunities	Threats
Fixa kompilator	Snabb installation, utförlig testkod	Mycket diskutrymme krävdes; kompilatorn rymdes inte på KTH:s 200 MB per användare	Inget krångel	Inga binära filer, ingår inte i OS
Kompilera HelloWorld	Kompilerade hela vägen och ger körbar fil, skönt att slippa virtuell maskin	Konstiga programnamn som inte känns genomtänkta	Kompilatorn kändes lik gcc. Går också att göra via hemsidan	Konstiga kommandon
Wiki: HTTP-server	Smart med input/output för funktioner. Rätt så elegant när man väl har satt sig in. Många egenskaper man längtat efter ingår i språket	Svårt att se skillnad på måsvingar och vanliga parenteser. Inte bra att man ska deklarera funktioner utan att strukturera upp. Pekardeklarationen med receivers kändes inte logisk. Inte lätt att komma ihåg när man ska använda rätt pekartecken (& och $*$)	Man förstår hur de har tänkt med språket. Det gick bra och känns naturligt att utveckla i, intuitivt för programmerare. Gick förvånansvärt fort att lära sig.	Känns svårt vid en första blick. Fick fel i kompilatorn, även om de rättades till. Svårt att hitta felet i koden trots att den skrevs av. Typdeklarationen []Byte, byte slices, kändes inte naturlig
Wiki: HTTP-server	Användbart att det går så fort att fixa en simpel server	-	Syntaxen känns igen från Perl/Matlab (r.URL.Path[1:])	HandleFunc känns underlig Svårt att förstå hur paket fungerar och om de motsvaras av klasser och hur deras funktioner ska användas

Pass	Strengths	Weaknesses	Opportunities	Threats
Wiki: Integrera (3) och (4)	Enkelt att integrera komponenter	Programmet kraschade under körning och gav svårtolkad debug-information bara för att en pekare användes fel	Inga nämnvärda svårigheter utöver krasch	Koden som Golang lagt upp fungerade inte. Kompilatorn som användes tycks ha uppdaterats så att vår kod inte fungerade
Wiki: Mallar i Go	Bra att templates finns inbyggt. Bra att slippa hårdkodning. "Vansinne" att inte ha mallar	Kunde varit mer trivialt. Lätt att skriva fel på parsersträngar och få fel som upptäcks först under körning	-	Ser inte enkel ut, svårt att motivera att varje rad behövs. Body|html parsades på ett sätt som inte var uppenbart
Wiki: Felhantering och problem	Bra att felfunktioner finns inbyggda i http-paketet	Modellen för felhantering kommer inte passa inte alla sorters tillämpningar	Lätt att lägga in felhantering	-
Wiki: Cachning	Snygg och koncis deklaration. Bra att cachad information läggs in i arrays	Varför man ska skriva make() är inte uppenbart	-	Svår syntax som vi inte hade kommit på själva. Syntax som liknar haskell, språk inte alla är vana vid
Wiki: Säkerhetsluckor	Elegant att man förkompilera regex-uttryck redan i början	-	Lätt att lägga in regex-uttryck i vanliga variabler. Regex-uttrycken används som i andra språk	-
Wiki: Återanvändning av kod	Bra att man slipper reflektion som man har i java. Praktiskt med anonyma funktioner	Koden kan bli mer svårläst snabbt grötig i statisk typning då hela funktionssignaturen skickas med.	Kan vara ett enklare alternativ till funktionspekare	Funktionspekare i C är svårt att använda, samma tros gälla för returnering av funktioner

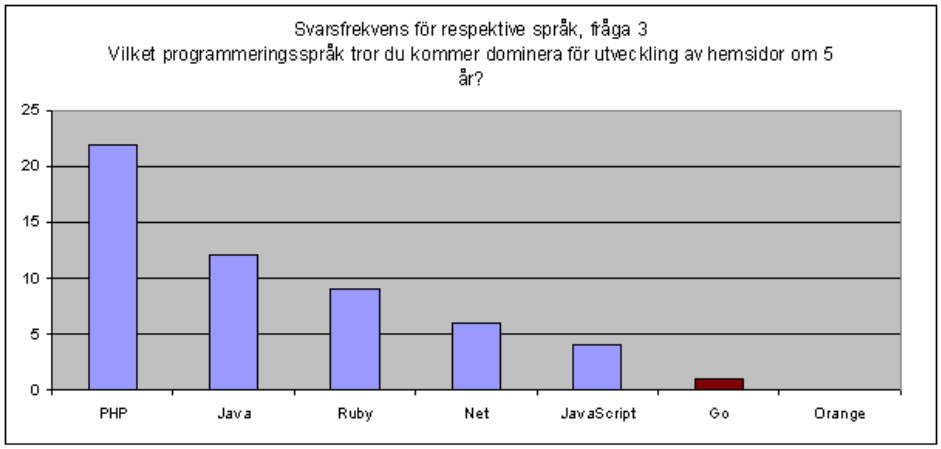
4.3 Enkätundersökning



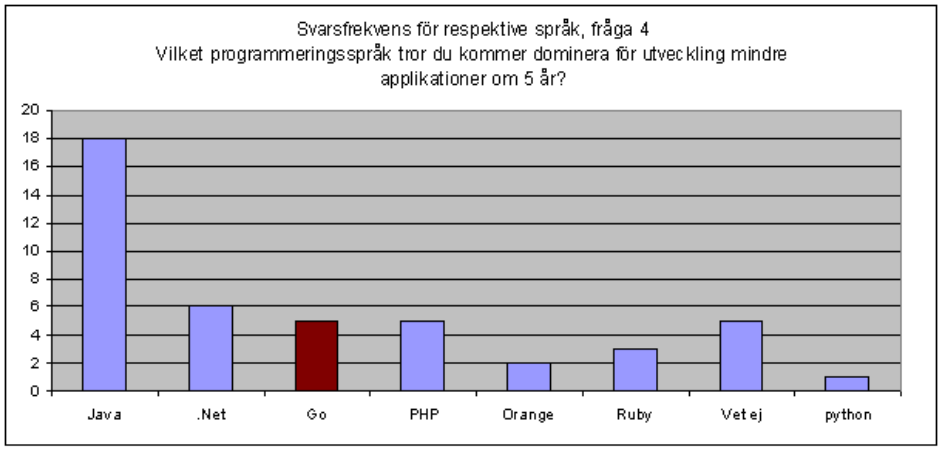
Figur 4.1. Svarsfrekvens för respektive språk, fråga 1: Vilket eller vilka av följande programmeringsspråk har du stött på? Antalet kryss per språk delat med totalt antal deltagare (50 personer)



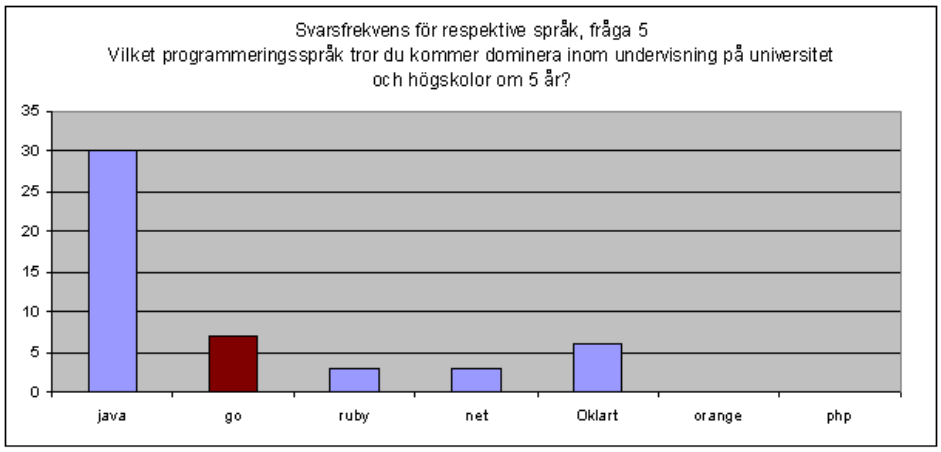
Figur 4.2. Svarsfrekvens för respektive språk, fråga 2: Vilket eller vilka av följande programmeringsspråk skulle du kunna tänka dig att lära dig inom en 5 års peroid? Antalet kryss per språk delat med totalt antal deltagare (50 personer)



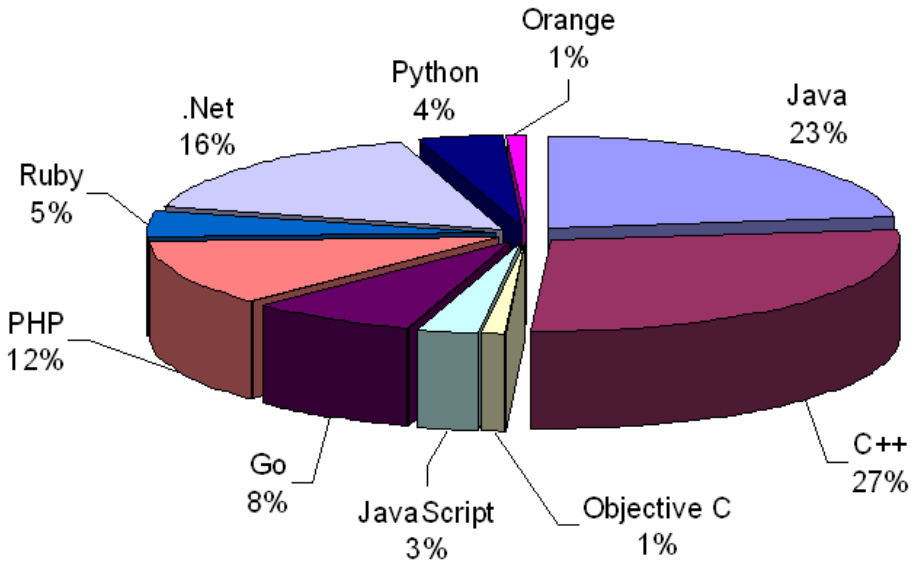
Figur 4.3. Svarsfrekvens för respektive språk, Fråga 3: Vilket programmeringsspråk tror du kommer dominera för utveckling av hemsidor om 5 år? Antalet kryss per språk, totalt 50 deltagare, fler än ett kryss kunde ges.



Figur 4.4. Svarsfrekvens för respektive språk, Fråga 4: Vilket programmeringsspråk tror du kommer dominera för utveckling mindre applikationer om 5 år? Antalet kryss per språk, totalt 50 deltagare, fler än ett kryss kunde ges.



Figur 4.5. Svarsfrekvens för respektive språk, Fråga 5: Vilket programmeringsspråk tror du kommer dominera inom undervisning på universitet och högskolor om 5 år? Antalet kryss per språk, totalt 50 deltagare, fler än ett kryss kunde ges.



Figur 4.6. Svarsfrekvens för respektive språk, Fråga 6: Vilka tre programmeringsspråk anser du vara mest värda att lära sig inom en 5 år? Cirkeldiagrammet visar den procentuella fördelningen av givna röster. Blanksvar utelämnade.

Kapitel 5

Analys

5.1 Tidsåtgång

De flesta uppgifterna gick snabbt att utföra men det måste hållas i åtanke att de flesta var mycket små och enkla. Mycket tid gick åt att arbeta med krascher och felsökning efter att fel uppstått. Hade Go haft en bra IDE och en mer stabil kompilator hade förmodligen färre problem uppstått.

5.2 Fördelar/nackdelar

5.2.1 Installation

Det uppskattades att det gick snabbt och relativt smärtfritt att kompilera och installera Go. Även om det inte fanns binära filer så är det inte helt otänkbart att flera programmerare är rutinerade i sådant. Vi bedömer också att många programmerare har en kunskapsnivå och erfarenhet som generellt överstiger vår. Däremot kan det vara ett hinder att programmet inte finns inlagt i operativsystemen när de levereras, som till exempel gcc gör i vissa Linux-distibutioner. Det fanns inte heller något enkelt sätt att automatiskt ladda ner Go-kompilatorn på de testdatorer vi använde oss av. Troligtvis påverkar det hur stor andel av de potentiellt intresserade som lyckas få igång en kompilator givet den lilla tid som vi tror att professionella programmerare har att lägga ner på att testa den uppsjö av programspråk som finns tillgängliga.

5.2.2 Kompilering

Go har mycket speciella kommandon för att kompilera och länka koden. Typiskt kör förmodligen de flesta programmen i en terminal eftersom ingen IDE finns tillgänglig för Go. Av den anledningen blir det viktigt att kommandona känns enkla och självklara. För att kompilera skriver man till exempel 8g code.go, 8l code.8 och ./8.out för vanliga 32 bitars pc-datorer, och där åttan ersätts av sexor för 64-bitars datorer. Varför just de kommandona valts som standard motiveras inte tydligt av de hemsi-

dor som rapporten innefattar. Programmet kraschade under körning vid ett tillfälle. Kompilatorn gav också felmeddelanden som var svårtolkade efter att enstaka bokstäver saknades. Dessa brister tyder på att kompilatorn är i ett utvecklingsstadium. Värre är att felen som uppkom var svåra att hitta i koden. Försöksprogrammerarnas fel var svåra att upptäcka eftersom koden såg normal ut.

5.2.3 Funktionsanrop

Funktionerna som beskrivs i wiki-projektet har deklarationen INDATA funktionsnamn UTDATA. Syftet med detta tros vara att visa hur flödet passerar genom en maskin med in- och utmatning. En annan intressant detalj som är påtagligt använd i Go är pekare som i allmänhet dyker upp på flera olika sammanhang. Språket påminner om C:s pekare. Det hade varit önskvärt med en enklare notation än $*$ och & som C använder för att teckna värdet av en pekare (det pekaren pekar på) respektive adressen av en variabel. Vidare kan funktioner returnera funktioner. Möjligen är den möjligheten tänkt att ersätta funktionspekare i C, som av uppsatsförfattarna bedöms ha förbättringspotential.

5.2.4 Paket

Avsaknaden av klasser började märkas i takt med att koden växte. Att funktioner deklareras globalt, som i till exempel C, kan vara en potentiell nackdel då detta är en positiv egenskap som lagts till i mer moderna språk såsom C++. Å andra sidan var "paketen", någonting som liknar vanliga importerade bibliotek som finns i de flesta programmeringsspråken, en möjlighet att strukturera upp koden. Komponentbaserad programmering borde vara en lämpligare paradigm för Go än att dela upp arbetet i klasser som man traditionellt skulle göra. Om Go är ett bra språk för komponentbaserad programmering eller inte går inte att ge ett säkert svar på med hänsyn till arbetets omfattning. Däremot gav försöksprogrammeraren många positiva kommentarer om att de funktioner som importerades med paketen var användbara. Det som gör Go bra för komponentbaserad programmering är förmodligen att paketen är skrivna för att användas som komponenter. Till exempel var http-paketet användbart. En smart detalj var att uttryck som användes med paketet var anpassade så att internetprogrammerare kan känna igen sig.

5.2.5 Datastrukturer

Pekarna som finns i koden uppfattades stundvis som svåra och omotiverade. Förväntningarna på detta område var stora men de kodstycken som gavs som exempel var inte alltid lätta att förstå utan förklarande text. Ett alternativ till att använda pekare i vanlig, löpande kod kunde ha varit att anpassa språket så att pekare bara måste användas i tillämpade sammanhang som till exempel minneshantering och filhantering, områden som pekare är skapta för.

5.2.6 Go som http-server

Det ansågs vara enkelt att bygga en HTTP-server med hjälp av Go:s inbyggda http-paket. De "handlers" som används för att visa olika typer av sidor var lätta att använda samtidigt. Förmodligen är det också bra ur parallellprogrammeringssynpunkt att skapa flera olika handlers för att skapa separat kod för varje funktion som servern är tänkt att ha. Däremot uppfattades det exempel som Golang gav som rörigt i och med att man måste gå in på en subadress för att kunna använda en viss handler. Att ladda in paketet och skapa en HTTP-server uppfattades inte som svårt tack vare att paketet i egenskap av komponent redan hade de självklara funktionerna färdiga.

5.3 Enkätsvar

Enkäten talar för att Go inte kommer att vara ett dominerande språk i framtiden. Däremot bedömer enkätsvararna att Go kommer att vara större än flera andra mindre vanliga språk såsom Lisp, Objective C och Pascal. Värt att notera är att Go inte ofta tros vara dominerande för utveckling för hemsidor, vilket språkmakarna avsedde. Desto fler tror att Go är relativt lämpligt för utveckling av mindre applikationer och undervisning på universitet och högskolor.

Kapitel 6

Diskussion

Oavsett om ett språk ligger har tekniska fördelar i form av produktivitet, prestanda eller dylikt verkar onekligen en förvånansvärt stor del av framgångarna stå och falla med faktorer som ligger utanför det tekniska området. Någon djupare analys av det marknadsföringsvärde en tillämpning inom ett större kommersiellt projekt har för ett språk går det inte att blunda för att detta är av betydande karaktär. Man kan fråga sig vad som är mest värdefullt att satsa på, marknadsföring eller teknologisk kvalité. Troligtvis hade inte författarna kommit i kontakt med språket om det inte vore för det fokus på marknadsföring som upplevts under projektet.
Golang.org lyckades inte heller motivera varför ett nytt språk skulle behövas. Till exempel så kan C++, som är ett mycket senare språk och är mer av ett högnivåspråk än C, användas tillsammans med C. Det av försökspersonerna uppskattade http-paketet som var lätt att använda skulle kanske kunna integreras i andra språk. Vidare är det viktigt att ifrågasätta vem som har intresse av att forska om nytillkomna språk. Kanske är forskningen mer intressant för utvecklare än för framtida programmerare. Denna undersökning kartlagde fler faktorer som är till nytta för den tidigare kategorin än för den senare.
Med ovanstående iakttagelser tagna i beaktning verkar det till de närmaste omöjligt att dra några konkreta slutsatser ifrån en förhållandevis mindre undersökning av ämnet. Även inom mer rigorösa undersökningar, kommer slutsatser på allt utom väldigt kortsikt riskera att bli rent spekulativa.

Kapitel 7

Slutsats

Go visade sig vara ett ojämnt språk som delvis misslyckats med sin ambition att vara enkelt. Det var inte enkelt att sätta sig in i kod och de nya sätt att programmera på som introducerades. Språket uppfattades inte som lätt att vänja sig vid. Språket gav ofta intryck av att ha härmat efter andra kända språk men det fick inte försöksprogrammerarna att känna igen sig i Go. Även om språket i sig ligger tekniskt i framkant, saknas fortfarande stöd för Windows och någon IDE är i nuläget inte heller tillgänglig. Trots detta pekar enkäterna på att det finns ett stort intresse för Go relativt språkets tidiga stadium i utvecklingen. Men för att språket skall komma igång krävs sannolikt ett större kommersiellt intresse uppstår. Några konkreta slutsatser gällande språkets framtid är i nuläget svårt att säkert uttala sig om.

Kapitel 8

Rekommendationer för fortsatta studier

För att få en mer realistisk uppfattning om ett språks framtidspotential krävs studier som undersöker inte bara de tekniska aspekterna av ett språk, utan även de effekter som uppkommer av ett politiskt och juridiskt spel mellan de större företagen på världsmarknaden.

Språkets prestanda kräver mer omfattande studier. Om prestandavinsten skulle visa sig vara låg är inte Go ett intressant språk för prestandakrävande program.

Kapitel 9

Felkällor

9.1 Enkätundersökningen

I enkätundersökningen ingick endast studeranden vid datateknik på KTH. Samtliga gick sitt första år. Eftersom undersökningen genomfördes i pausen på en föreläsning föll de som inte valde att närvara på den specifika föreläsningen bort. Även homogeniteten bland de undersökta är problematisk. Alla programmerare är inte studerande på datateknikprogrammet, så den uppmätta datan måste inte exakt motsvara den verkliga uppfattningen bland programmerare generellt. De undersökta kan vidare sakna praktisk erfarenhet av programmering och har sannolikt i låg utsträckning varit med i något större mjukvaruutvecklingsprojekt. Det är svårare att dra några generella slutsatser av en undersökning baserad på enbart oerfarna studenter. Speciellt då många av dem uppger att de aldrig stött på eller hört talas om större programspråk som PHP eller Ruby, måste deras spekulationer om Go:s framtidsutsikter betraktas som opålitliga och säger inte mycket om hur vana programmerare uppfattar framtidsutsikterna. Detta problem kvarstår dock oavsett vilken årskurs enkäten vänder sig till, i och med att alla är studenter och därmed inte kan förväntas ha någon större praktisk erfarenhet. Bland de sista årskurserna har studenter redan valt inriktning och är då troligen fokuserade på en viss typ av programspråk. För ett mer pålitligt resultat hade en mer omfattande undersökning behövts göras. Förslagsvis genom att undersöka ett bredare urval.

9.2 Wikipedia

Wikipedia är en tveksam källa för information om en kommersiell produkt som utvecklas av ett stort företag, som Go är. Därför har vi valt att begränsa källanvändningen till trivial information som ingen sannolikt har intresse av att vinkla eller information som återfunnits även på andra källor.

Kapitel 10

Källförteckning

http://en.wikipedia.org/wiki/Component-based_software_engineering
Hämtad 2011-04-13
http://en.wikipedia.org/wiki/Integrated_development_environment
Hämtad 2011-04-09
http://en.wikipedia.org/wiki/Go_%28programming_language%29
Hämtad 2011-04-01
http://golang.org/doc/go_faq.html
Hämtad 2011-04-13
http://sv.wikipedia.org/wiki/Dotnet_Framework
Hämtad 2011-04-01
http://golang.org
Hämtad 2011-04-01
Dahmström, Från datainsamling till rapport, 3:e upplagan, 2000, Studentlitteratur AB, Lund
Thurén, Vetenskapsteori för nybörjare, 2:a upplagan, 2007, Liber, Malmö

10.1 Insamlad data

http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
Hämtad 2011-04-03

Kapitel 11

Bilaga A - Enkät

Framtidens Programmeringsspråk

Denna enkät undersöker graden av kännedom och tilltro för olika programmeringsspråk.

Program/årskurs (t ex D-10) ☐

Vilket eller vilka av följande programmeringsspråk har du stött på?

☐ C++	☐ Perl	☐ Lua	☐ Visual Basic .Net
☐ Python	☐ Ruby	☐ Go	☐ Fortran
☐ PHP	☐ Delphi	☐ RPG (OS/400)	☐ Pascal
☐ C#	☐ Lisp	☐ Scheme	☐ ADA
☐ Visual Basic	☐ Orange	☐ Objective-C	☐ JavaScript

Vilket eller vilka av följande programmeringsspråk skulle du kunna tänka dig att lära dig inom en 5 års peroid?

☐ C++	☐ Perl	☐ Lua	☐ Visual Basic .Net
☐ Python	☐ Ruby	☐ Go	☐ Fortran
☐ PHP	☐ Delphi	☐ RPG (OS/400)	☐ Pascal
☐ C#	☐ Lisp	☐ Scheme	☐ ADA
☐ Visual Basic	☐ Orange	☐ Objective-C	☐ JavaScript

Vilket programmeringsspråk tror du kommer dominera för utveckling av hemsidor om 5 år? (Nej, html är inget programmeringsspråk.)

☐ .Net ☐ Orange ☐ PHP ☐ Java ☐ Go ☐ Ruby Annat ☐

Vilket programmeringsspråk tror du kommer dominera för utveckling mindre applikationer om 5 år?

☐ .Net ☐ Orange ☐ PHP ☐ Java ☐ Go ☐ Ruby Annat ☐

Vilket programmeringsspråk tror du kommer dominera inom undervisning på universitet och högskolor om 5 år?

☐ .Net ☐ Orange ☐ PHP ☐ Java ☐ Go ☐ Ruby Annat ☐

Vilka tre programmeringsspråk anser du vara mest värda att lära sig inom 5 år?
(Kan inkludera ej listade språk eller språk du redan kan)

1 (Mest värdefullt) ☐

2 ☐

3 ☐

Kapitel 12

Bilaga B - Programkod i Go

Koden är skapad av samt tillhör Golang.org och användes som pedagogisk bas för försökspersonerna. Golang.org gav också en steg-för-steg guide som tillhörde koden. Filer och information finns på:
http://golang.org/doc/codelab/wiki/

12.1 wikiserver.go

```
package main
import (
http
io/ioutil
os
regexp
template
)
type Page struct {
Title string
Body []byte
}
func (p *Page) save() os.Error {
filename := p.Title + ".txt"
return ioutil.WriteFile(filename, p.Body, 0600)
}
func loadPage(title string) (*Page, os.Error) {
filename := title + ".txt"
body, err := ioutil.ReadFile(filename)
if err != nil {
return nil, err
}
```

```
return &Page{Title: title, Body: body}, nil
}
func viewHandler(w http.ResponseWriter, r *http.Request, title string) {
p, err := loadPage(title)
if err != nil {
http.Redirect(w, r, "/edit/"+title, http.StatusFound)
return
}
renderTemplate(w, "view", p)
}
func editHandler(w http.ResponseWriter, r *http.Request, title string) {
p, err := loadPage(title)
if err != nil {
p = &Page{Title: title}
}
renderTemplate(w, "edit", p)
}
func saveHandler(w http.ResponseWriter, r *http.Request, title string) {
body := r.FormValue("body")
p := &Page{Title: title, Body: []byte(body)}
err := p.save()
if err != nil {
http.Error(w, err.String(), http.StatusInternalServerError)
return
}
http.Redirect(w, r, "/view/"+title, http.StatusFound)
}
var templates = make(map[string]*template.Template)
func init() {
for _, tmpl := range []string{"edit", "view"} {
templates[tmpl] = template.MustParseFile(tmpl+".html", nil)
}
}
func renderTemplate(w http.ResponseWriter, tmpl string, p *Page) {
err := templates[tmpl].Execute(w, p)
if err != nil {
http.Error(w, err.String(), http.StatusInternalServerError)
}
}
const lenPath = len("/view/")
var titleValidator = regexp.MustCompile("^[a-zA-Z0-9]+$")
func makeHandler(fn func(http.ResponseWriter, *http.Request, string)) http.HandlerFunc {
return func(w http.ResponseWriter, r *http.Request) {
title := r.URL.Path[lenPath:]
if !titleValidator.MatchString(title) {
http.NotFound(w, r)
return
}
fn(w, r, title)
```

```
}
}
func main() {
http.HandleFunc("/view/", makeHandler(viewHandler))
http.HandleFunc("/edit/", makeHandler(editHandler))
http.HandleFunc("/save/", makeHandler(saveHandler))
http.ListenAndServe(":8080", nil)
}
```

12.2 edit.txt

```
<h1>Editing {Title}</h1>
<form action="/save/{Title}" method="POST">
<div><textarea name="body" rows="20" cols="80">{Body|html}</textarea></div>
<div><input type="submit" value="Save"></div>
</form>
```

12.3 view.txt

```
<h1>{Title}</h1>
<p>[<a href="/edit/{Title}">edit</a>]</p>
<div>{Body}</div>
```