

Skriptspråk: framtidens programspråk

Christian Davén
9E1363 Muntlig och skriftlig framställning
KTH

14 april 2004

Sammanfattning

Framöver kan skriptspråk komma att spela en allt viktigare roll inom systemutveckling. En kort historisk genomgång följs av en diskussion kring deras egenskaper. Slutligen redovisas en empirisk jämförelse mellan skriptspråk, representerade av bland andra Perl och Python, och traditionella systemprogramspråk som C och Java.

1 Introduktion

Skriptspråk har använts under mer än tre decennier och har under de senaste åren blivit allt mer betydelsefulla inom industrin. För programvaruingenjörer blir det allt mer viktigt att leverera ett funktionsdugligt system så snabbt som möjligt, och skriptspråken tros kunna underlätta det arbetet jämfört med de traditionella systemprogramspråken. Idag finns ett flertal skriptspråk som anses vara fullfjädrade programspråk och som används för utveckling av de flesta typer av system. Detta tillsammans med skriptspråkens övriga positiva egenskaper öppnar för att de ska komma att användas än mer i framtiden.

Denna rapport försöker visa dessa programspråks kvaliteter och höja deras status från nybörjarspråk till fullt användbara även i kommersiella projekt. Först ges en kort bakgrund, därefter följer en diskussion kring några av programspråkens egenskaper. Sedan redovisas en empirisk jämförelse och slutsatser.

2 Bakgrund

2.1 Programspråkens utveckling

De mest grundläggande programspråken är assemblerspråken. De består av korta kommandon som översätts direkt till maskinkod och är därmed helt beroende av plattform.

Efter att högnivåspråk som FORTRAN utvecklats kunde programmeraren skriva mer kompakt och lättförståelig kod. Kompilatorn översatte sedan denna till assemblerkod för att kunna förstås av datorn, och i genomsnitt ersattes varje rad FORTRAN-kod med två till tre rader assemblerkod (Jones 1996). Senare kom C som är ungefär lika uttrycksfullt. Med C följer också C Standard Library som innehåller de mest grundläggande funktionerna och algoritmerna för bland annat matematiska operationer och stränghantering.

C++ lanserades som en vidareutveckling av C och är ännu mer kompakt. Ett av målen med att skapa språket var att höja abstraktionsnivån på programmeringen så att datalogiska koncept kunde bli synliga direkt i koden (Allison 1996; Venners 2004). Då kan programmeraren skapa program med hjälp av ”grövre byggstenar” istället för att använda de minsta beståndsdelarna som bygger upp assemblerkod. Det resulterar i mindre kod och enklare underhåll (Venners 2004). I språket ingår också Standard Template Library, ett mycket större bibliotek än C Standard Library med kod som kan användas för att förenkla vanliga uppgifter.

Ett decennium senare lanserade Sun Microsystems språket Java, starkt influerat av C++. Med språket kommer också Java API, ett ännu större bibliotek med kod för att förenkla vanliga uppgifter, som exempelvis nätverks- och gränssnittsprogrammering. Java reducerar utvecklingskostnaderna och snabbar upp inlärningskurvan jämfört med C och C++ (Sabharwal 1998).

C, C++, Java och deras släktingar kallas ibland systemprogramspråk för att skilja dem från skriptspråk.

2.2 Skriptspråken gör entré

De första skriptspråken kan sägas vara de som användes på de första Unix-systemen under 1970-talet. De kunde användas till att starta flera program med ett enda kommando och innehöll ofta villkorssatser för att under vissa omständigheter utföra ett annat kommando. Så småningom dök Awk upp, anpassat speciellt för att analysera och söka i textmassor med reguljära uttryck (eng. *regular expressions*).

Perl, som kom 1987, var starkt influerat av Awk och följdes snart av Tcl och Python. Genom åren har de mognat från att vara enkla skriptspråk till att vara fullfjädrade programspråk och fått sällskap av allt fler, exempelvis PHP och Ruby. Idag går det i de flesta fall att skriva samma program i Perl som i C, med den fördelen att Perl-programmeraren blir klar snabbare (Dominus 1998). Även skriptspråken inkluderar bibliotek för att utföra vanliga uppgifter som att hantera nätverkskopplingar, databaser etc.

2.3 Dokumenterad användning av skriptspråk

På många håll har man inom industrin insett att skriptspråken har vissa fördelar gentemot systemprogramspråken. Det måste anses vara ett godkännande att användas i kommersiella system. Här är ett urval:

- Human Genome Project (HUGO), som kartlade hela den mänskliga arvsmassan under 1990–2003, använde Perl för att hantera och jämföra enorma mängder genetisk data (Dominus 1998).
- Astrazeneca använder ett system skrivet i Python för att underlätta forskarnas arbete med att ta fram nya läkemedel (Boyer, Dalke & Bruneau 2003).
- Lufthansas onlinesystem för biljettbokning, betalning med mera, är skrivet i PHP (Zend Technologies 2003).
- Den svenska Premiepensionsmyndigheten använder Perl för att hantera hela det nya premiepensionssystemet med över fem miljoner personers fondportföljer (Stephenson 2001).

2.4 Framtidens utmaningar

Enligt Sommerville (2001) är två av de stora utmaningarna för 2000-talets programvaruingenjörer att kunna:

- leverera samma system till flera olika plattformar
- korta ned dagens leveranstider utan att tumma på kvaliteten.

Dessa två utmaningar måste övervinnas av utvecklingsföretagen för att kunna möta den hårdnande konkurrensen. Med dagens språk och metoder är det svårt, men nedan diskuteras bland annat hur skriptspråken kan bidra till en lösning.

3 Diskussion

3.1 Statisk och dynamisk typning

All data i ett program är av en viss typ eller sort, exempelvis ett heltal eller en sträng. Om en och samma variabel vid olika tidpunkter kan innehålla data av olika typ säger man att det programspråket har dynamisk typning, eller är dynamiskt typat. Motsatsen är statisk typning, att varje variabel ges en permanent typ och endast kan innehålla data av denna typ.

De maskinnära assemblerspråkens variabler pekar endast på platser i minnet eller på register, och säger ingenting om datans typ. Vid ett tillfälle kan en minnesadress tolkas som en sträng, vid ett annat som ett heltal. Assemblerspråken är dynamiskt typade och det är helt upp till programmeraren hur värden ska tolkas.

När högnivåspråken skulle göra programmeringen enklare och mer felfri införde man statisk typning. Detta gör att programmeraren kan se direkt vilken typ som förväntas användas i exempelvis funktionsanrop, vilket gör programmen mer lätthanterliga. Kompilatorn kan också fånga upp många programmeringsfel som exempelvis addition av en sträng och ett heltal, vilket ger färre buggar. Dessutom kan kompilatorn optimera den färdiga koden tack vare att den vet av vilken typ varje variabel är (Ousterhout 1998).

Skriptspråken har däremot återgått till assemblerspråkens dynamiska typning för att snabba upp utvecklingen av programvara. Fördelarna är dels att det går snabbare att skriva kod när programmeraren inte behöver fundera över vilken typ som passar bäst för en given uppgift, dels att det går snabbare att ändra i koden när programmeraren inte behöver ändra typen för variabler på flera ställen (Martin 2003).

En metod för att utveckla programvara är Extreme programming (XP), som blivit allt mer populär de senaste åren. XP bygger till stor del på kontinuerlig testning av all kod som skrivs. Dessa tester hittar även typfel och gör att kompilatorns test av variabeltyper blir onödig. Eftersom detta var en stark anledning till den statiska typningen, och eftersom dynamisk typning gör programmerarens arbete lättare, är XP tillsammans med ett dynamiskt typat programspråk en lämpligare lösning än att använda systemprogramspråken med traditionella metoder (Martin 2003).

3.2 Uttrycksfullhet

I genomsnitt motsvarar varje kodrad skriven i ett skriptspråk två till tre rader Javakod och sex rader C-kod (Jones 1996). Skriptspråken är alltså mer uttrycksfulla än systemprogramspråken, något som kan ses som ett uttryck för högre abstraktionsnivå. Istället

för att skyffla data mellan register och anropa avbrott, som man gör i assembler, skriver man mer i klartext vad man vill göra. Ett exempel är att avsluta ett program: då skriver man ofta "exit" i högnivåspråk, medan man i assemblerspråk måste använda kod som är till förväxling lik all annan. Högre abstraktionsnivå gör koden enklare att producera och överblicka.

Detta gör att nybörjare snabbare kan skriva program i mer uttrycksfulla språk eftersom mindre detaljkunskaper krävs. Eftersom skriptkoden översätts till generellt optimerad assemblerkod behövs heller inte lika mycket kunskap om optimering, som dessutom ofta är plattformsspecifik. Dessutom finns en sedan länge accepterad tumregel om att en programmerares produktivitet i antal rader är oberoende av programspråk (Prechelt 2000). Den regeln säger således att programmeraren som använder ett skript-språk är mer än dubbelt så produktiv som Java-programmeraren. Dessa siffror bekräftas nedan under *Empiriska jämförelser*. Högre uttrycksfullhet gör å andra sidan också att programmeraren har mindre kontroll över hur den kompilerade koden kommer se ut och har mindre möjligheter att själv optimera den för prestanda.

3.3 Portabilitet

Portabilitet behövs eftersom många program idag behöver kunna fungera på flera olika plattformar och för att öka utvecklingsföretagens möjligheter till att återanvända kod i program för flera kunder. Kod som skrivits för en plattform kan då enkelt flyttas över till och användas på en annan. Exempelvis kompileras Javakod till ett plattformsoberoende pseudo-assemblerspråk. För att köra programmen krävs en tolk kallad Java virtual machine (JVM), som ser till att maskinspecifik kod körs. Överallt där det finns en JVM kan samma kod användas, oavsett om det är en handdator, webbserver eller arbetsstation.

Tvärt emot Java kompileras C-kod till assemblerkod för en specifik plattform och går endast att köra där. För varje plattform ett program ska fungera på krävs åtminstone ännu en kompilering från programkod. I många fall går det att direkt kompilera samma kod för många olika plattformar, men ofta är själva koden plattformsspecifik och kräver därför mer arbete för att kunna porteras till en annan plattform.

Skriptspråken använder en lösning som liknar Javas, men koden kompileras inte till pseudo-kod utan behålls i ursprungligt skick. När programmen sedan körs på en plattform är det en tolk som översätter koden. Ett program kan köras på varje plattform där det finns en tolk för språket, och skriptspråken har i detta avseende en enorm popularitet då det finns tolkar till ett mycket stort antal plattformar.

3.4 Prestanda

Eftersom systemprogramspråken teoretiskt sett kan åstadkomma snabbare och mer minnessnåla program, är dessa fortfarande lämpligare att använda där man har mycket höga krav på prestanda. Exempel på sådana domäner är grafik- eller beräkningsintensiva spel och program.

4 Empiriska jämförelser

Prechelt (2000) har jämfört prestanda hos 80 program skrivna i sju olika programspråk, som alla löste samma uppgift. Programmen läste in en ordlista från en textfil, en lista med telefonnummer från en annan fil, ersatte varje telefonnummer med ord i ordlistan

utifrån vissa regler och visade slutligen resultatet. Alla program var skrivna av olika programmerare.

Språken var dels systemprogramspråk; C, C++ och Java, dels skriptspråk; Perl, Python, Rexx och Tcl. Till viss del kommer jämförelserna vara mellan systemprogram- och skriptspråken som grupper och till viss del mellan C och C++ som en grupp, Java ensamt och skriptspråken som en grupp. Detta för att Java ibland utmärker sig bland systemprogramspråken.

4.1 Snabbhet

Tolkningen som läggs fram är att medelvärdet på exekveringstiden inte varierar särskilt mycket mellan programspråken. I varje språk utom Tcl och Rexx har det skrivits program som körs på under tio sekunder, samtidigt som det i varje språk har skrivits program som körs på mer än 16 minuter. Dock visas statistiskt att C och C++ ger 30% snabbare program än skriptspråken och 20% snabbare än Java. Mellan de två sistnämnda kan dock ingen statistiskt säkerställd skillnad visas upp.

Skillnaden mellan en duktig och en dålig programmerare är absolut störst för C och C++, därefter Java. För skriptspråken är skillnaden mycket mindre; nybörjaren skriver nästan lika effektiva program som experten.

4.2 Minneskrav

När det gäller minneskrav för programmen är C och C++ bäst, Java absolut sämst och skriptspråken mitt emellan. Men även här är variationerna stora: de sämsta C-programmen är sämre än de flesta skriptspråksprogrammen. I genomsnitt kräver en Java-implementation dubbelt så mycket minne som en skriptspråks-dito, som i sin tur kräver dubbelt så mycket minne som ett program implementerat i C eller C++.

Variationerna mellan duktiga och dåliga programmerare är återigen mindre för skriptspråken, i synnerhet för Python. I denna aspekt ger C++ och Java störst variationer.

4.3 Utvecklingstid

De absolut kortaste utvecklingstiderna var de för skriptspråken, därefter C och C++ och sist Java. Detta förklaras med att Java-programmerarna bör ha varit mindre erfarna då språket vid testtillfället (1997–98) var förhållandevis nytt. Undersökningen visar att utveckling i skriptspråk går dubbelt så fort som i systemprogramspråk, även om duktiga programmerare i varje språk kan utveckla program snabbare än sämre programmerare i de andra språken.

5 Slutsats

På samma sätt som C höjde abstraktionsnivån från assemblerprogrammeringen höjer skriptspråken densamma ytterligare en nivå. Tack vare detta är det enklare att skriva och överblicka program skrivna i skriptspråk, vilket ökar produktiviteten. Skillnaderna mellan en bättre och en sämre programmerare minskar också i och med att en större del av arbetet utförs av programtolken.

Dynamisk typning, som används i skriptspråken, ökar utvecklingshastigheten ytterligare – till kostnaden av att testningen måste vara noggrann, vilket den ändå är om

utvecklarna använder sig av XP. Kod skriven i skriptspråk är också mer portabel än den skriven i C och C++ och jämförbar med Java. Detta oberoende av plattform gör att återanvändandet kan öka och att samma system direkt kan användas på flera plattformar, vilket kan öka utvecklarnas intäkter per skrivet program.

Enligt en empirisk undersökning av skillnaderna mellan implementationer i olika programspråk är skriptspråken ungefär likvärdiga Java i snabbhet, men med halverade minneskrav. C och C++ ger något snabbare program som kräver endast hälften så mycket minne som skriptspråken. Dock går det ungefär dubbelt så fort att utveckla program i skriptspråk jämfört med systemprogramspråken. Det finns domäner där systemprogramspråken kommer att leva kvar åtminstone ett tag till. Dessa är de där snabbheten är viktigare än utvecklingstid och underhåll.

Som allt annat kommer programmeringen fortsätta att utvecklas, frågan är bara hur. Det vore naivt att tro att Java alltid kommer vara det mest populära programspråket (TIOBE Software 2004), men vem kommer ta över tronen härnäst? Ett rimligt antagande är att det blir något av skriptspråken, men inte nödvändigtvis ett redan existerande. Hur snabbt språkskiftet kommer ske beror till stor del på information och marknadsföring; att övertyga utvecklarna om att det finns ett språk bättre lämpat för att möta 2000-talets utmaningar.

Referenser

- Allison, Chuck (1996). C++: The Making of a Standard Journey's End: An Interview with Bjarne Stroustrup [Elektronisk]. *C/C++ User's Journal*, 1996:10. Tillgänglig: <http://www.research.att.com/~bs/cuj_interview.html> [läst 2004-03-29].
- Boyer, S., Dalke, A. & Bruneau, P. (2003). *AstraZeneca Uses Python for Collaborative Drug Discovery* [Elektronisk]. Tillgänglig: <<http://pythonology.org/success&story=astra>> [läst 2004-03-29].
- Dominus, Mark-Jason (1998). Perl: Not Just for Web Programming. *IEEE Software*, vol. 15:1, ss. 69–74.
- Jones, Capers (1996). *Programming Languages Table 8.2* [Elektronisk]. Tillgänglig: <<http://www.theadvisors.com/langcomparison.htm>> [läst 2004-03-29].
- Martin, Robert C. (2003). Are Dynamic Languages Going to Replace Static Languages? [Elektronisk]. *Uncle Bob's Software Craftsmanship Corner*, 26 april 2003. Tillgänglig: <<http://www.artima.com/weblogs/viewpost.jsp?thread=4639>> [läst 2004-03-29].
- Ousterhout, John K. (1998). Scripting: Higher Level Programming for the 21st Century. *IEEE Computer*, vol. 31:3, ss. 23–230.
- Prechelt, Lutz (2000). An Empirical Comparison of Seven Programming Languages. *IEEE Computer*, vol. 33:10, ss. 23–29.
- Sabharwal, Chaman L. (1998). Java, Java, Java. *IEEE Potentials*, vol. 17:3, ss. 33–37.
- Sommerville, Ian (2001). *Software Engineering*, sjätte upplagan. Addison-Wesley.
- Stephenson, Ed (2001). *Perl Runs Sweden's Pension System* [Elektronisk]. Tillgänglig: <http://perl.oreilly.com/news/swedishpension_0601.html> [läst 2004-03-29].
- TIOBE Software (2004). *TIOBE Programming Community Index for March 2004* [Elektronisk]. Tillgänglig: <<http://www.tiobe.com/tpci.htm>> [läst 2004-03-29].

Venners, Bill (2004). *Abstraction and Efficiency: A Conversation with Bjarne Stroustrup* [Elektronisk]. *Artima Articles*, 16 februari 2004. Tillgänglig: <<http://www.artima.com/intv/abstreffi.html>> [läst 2004-03-29].

Zend Technologies (2003). *Lufthansa Selects PHP and Zend for Mission Critical Online eTicketing Operations* [Elektronisk]. Tillgänglig: <<http://www.zend.com/news/zendpr.php?id=60>> [läst 2004-03-29].