DD1366 Programmeringsparadigm X2

Erik Smit, Hugo Larsson Wilhelmsson
esmit@kth.se, hugolw@kth.se
20020924-2437, 20020921-1234

April 2024

Innehåll

1 COBOL **3**

2 Smalltalk **3**

3 Erlang **3**

4 PHP **4**

5 Clojure **4**

6 Rust **4**

7 Slutlig reflektion **5**

1 COBOL

Det imperativa paradigmespråket COBOL var interessant att programmera i! Överlag var syntaxen ganska lättförståelig och koden var hyfsat lätt att läsa. Utöver att språket var tydligt så finns det inte så mycket mer gott att säga om det. Nackdelarna med COBOL är att det finns brist på moderna funktioner och verktygsstöd samt den begränsade flexibiliteten i språket. Detta gjorde det ganska jobbigt och problematiskt att implementera denna simpla algoritm i språket. För ännu större/avancerade program lär problemen bara öka.

2 Smalltalk

Att programmera i Smalltalk har också varit en intressant upplevelse, och det finns både för- och nackdelar med språket.

En av de mest uppenbara fördelarna med Smalltalk är dess enkelhet och tydlighet. Syntaxen är konsekvent och lättläst. Vidare så är dess objektorienterade modell också ganska intuitiv och hyfsat väl anpassad för att utveckla större program.

Den största utmaningen som vi har mött med Smalltalk är att det är svårt att hitta hjälp och support. Dessutom är kompilatorns felmeddelanden ganska kryptiska och svårlästa. Detta gör debugging oerhört svårt och irriterande. Därför tror vi också att det kan vara jobbigt att konstruera större och mer avancerade program i detta språk.

3 Erlang

Erlang tillhör det funktionella paradigmet, vilket vi har programmerat mycket i de senaste månaderna. Vi tänkte därmed innan vi började med uppgiften att algoritmen borde vara relativt enkel att implementera i Erlang. Vi hade däremot fel eftersom syntaxen var skrämmande lik Prolog, vilket vi inte programmerat särskilt mycket i tidigare. Vi nämnde i vår rapport för X1 att vi tycker prolog är ett svårt språk, främst på grund av att det har en syntax som inte liknar särskilt många andra programmeringsspråk. Detta var en bidragande orsak till att Erlang var ganska svårt att arbeta med.

En av de starkaste fördelarna med Erlang enligt oss är att man kan dela upp programmet i flera olika funktioner väldigt lätt. Detta gjorde att vi kunde dela upp komplex kod i flera olika funktioner så det dels blev mer lättläsligt, dels blev lättare att implementera.

Överlag så tyckte vi det var givande att programmera i Erlang men vi tror att det är svårt att göra större program i språket. Anledningen till detta är främst att det inte är särskilt vanligt att använda Erlang idag (jämfört med många

andra språk) vilket därmed gör det svårt att veta var man ska vända sig om man behöver support.

4 PHP

Det imperativa/objektorienterade paradigmspråket PHP tyckte vi var det roligaste språket att implementera algoritmen i. Anledningen till detta är att vi tidigare hade hört att PHP ska vara ett ganska jobbigt språk att programmera i, vilket vi inte håller med om. Vi tyckte att PHP var likt Java, vilket vi programmerat mycket i tidigare. Detta resulterade i att det gick relativt snabbt att implementera algoritmen i språket.

Den största skillnaden jämfört med Java anser vi är att man behöver skriva $ innan varje variabel. För ett litet program, som i denna labben, var det inga större problem. Om man däremot ska programmera ett program som är flera tusen rader långt så kommer det medföra väldigt många $ vilket i sin tur kan göra programmet svårläst. Vi tror därmed att det är lätt att få syntaxfel om man kodar stora program i PHP, men vi tror att det är betydligt lättare än i de tidigare språken vi implementerat algoritmen i.

5 Clojure

Det har varit givande att programmera i det funktionella paradigmspråket Clojure och vi har märkt att det finns tydliga fördelar respektive nackdelar med språket.

Fördelarna med Clojure är bland annat att det finns ganska bra dokumentation och hjälp att få gällande syntaxen av språket, som för övrigt ändå är väldigt lättförstått. Vidare är det ett väldigt minimalistiskt språk vilket gör att koden blir ganska elegant. Dessutom har vi märkt att programmen man bygger i Clojure är väldigt robusta, delvis beroende på språkets fokus på immutability.

Däremot är det en del grejer som också har vart svårt i Clojure. Immutabilityn ställde till det lite gällande hur vi implementerade algoritmen och hur vi sparade värden, vilket inte var helt intuitivt. Dessutom tog det en del tid att vänja sig med användningen av paranteser samt hur matematiska uttryck representerades i Clojure. Men överlag går det nog ganska bra att bygga större och mer komplexa program med Clojure, särskilt program som körs på flera trådar då Clojure har inbyggd concurrency.

6 Rust

Multiparadigmspråket Rust var det sista språket vi implementerade algoritmen i. Ingen av oss hade kodat i Rust tidigare. Däremot så hade vi båda program-

merat i C++, vilket är ganska likt Rust. På grund av detta så gick det relativt smidigt att implementera algoritmen i språket.

Överlag så tycker vi att det var lätt att göra ett program som löser uppgiften, och som även är kort i radantal. Detta kan resultera i att det är smidigt att använda till stora program då man kanske inte behöver skriva lika många rader kod som i många andra språk.

En potentiell nackdel med Rust är att man behöver vara insatt i programmering för att förstå koden i jämförelse med till exempl Cobol där det mer explicit står vad som sker i programmet. Å andra sidan är det nog endast programmerare som kommer i kontakt med Rust så vi anser att detta inte är en nackdel som är avgörande för om man ska använda språket eller ej för att lösa en uppgift.

7 Slutlig reflektion

Vi tycker överlag att detta har varit en rolig labb, där vi framförallt har lärt oss vikten av att välja rätt språk när man löser en uppgift då svårigheten kan skilja mycket. Det snabbaste språket att impelmentera algoritmen i var för oss PHP, följt av Rust. Anledningen till detta är framförallt eftersom dessa språk är lika programmeringsspråk vi har erfarenhet av sedan tidigare.

Av de olika språken vi kodat i så tror vi att det främst är PHP, Rust och möjligtvid Clojure vi kommer använda oss av i framtiden, sålänge vi inte börjar arbeta med back-end på till exempel en bank där Cobol används. Anledningen till detta är att de andra språken inte används särskilt mycket idag och har ersatts av mer moderna och effektivare språk.