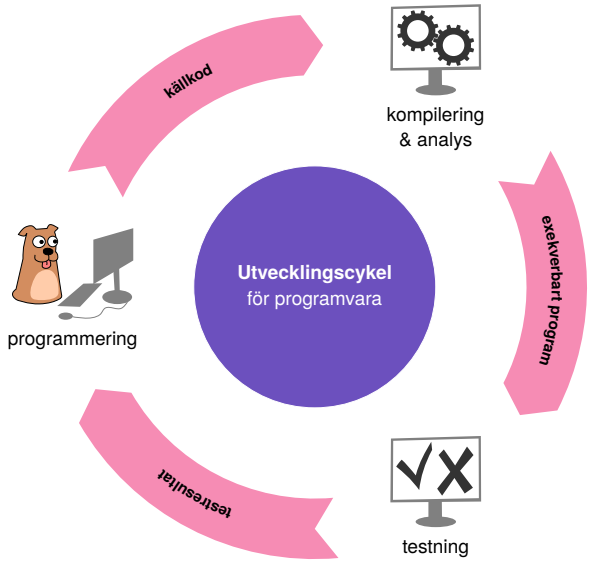
Utveckling av statisk analys

Statisk programanalys, eller kort och gott statisk analys, är av stor betydelse inom mjukvaruutveckling. Statisk analys är en samlingsterm för olika automatiserade analyser av datorprogram. Framför allt behövs statisk analys för att kompilera program till exekverbar maskinkod, så att de kan köras på en dator (eller mobiltelefon, till exempel). Statisk analys används också för att hitta och förhindra fel i program, samt för att optimera prestandan hos program. Med statisk analys kan man bland annat hitta säkerhetshål och förhindra att känslig data läcker ut ur ett program. Dessutom används statisk analys i *programmeringsverktyg*, alltså de verktyg en programmerare använder för att konstruera sina program. Till exempel kan statisk analys användas för att föreslå ändringar i koden (kodkomplettering och korrigering av enkla fel), eller för att låta programmeraren snabbt hoppa mellan definitioner och användningar av namn i koden.



Figur: Den vanliga utvecklingsprocessen för datorprogram. Under programmering och kompilering används statisk analys av programmeraren och kompilatorn.

Statisk analys är en form av mjukvara som ofta är komplicerad och tidskrävande att utveckla. Dessutom finns det ofta ett behov av att kunna bygga vidare på en befintlig statisk analys med nya egenskaper, till exempel om programmeringsspråket som analyseras uppdateras. Således är det önskvärt att utveckla statisk analys med en flexibel mjukvaruarkitektur, det vill säga på ett sätt som gör det möjligt att bygga på analysen utan allt för stor ansträngning.

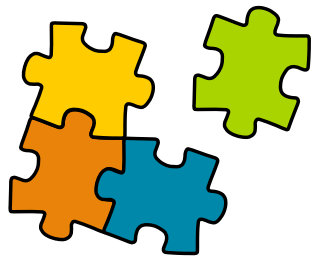
Givet att det finns några grundläggande analyser som är utvecklade med flexibel arkitektur så kan vi utveckla många nya viktiga analyser som använder de grundläggande analyserna som byggstenar. Detta sparar mycket tid och pengar.

Deklarativ programmering

För att uppnå en flexibel mjukvaruarkitektur kan vi använda oss av *deklarativ programmering*. Det innebär att programmeraren beskriver *vad* som skall beräknas, snarare än att exakt beskriva stegen som datorn tar för att beräkna det som behövs. Den främsta fördelen med deklarativ programmering, när det gäller att bygga flexibla program, är att det blir enkelt att dela upp ett deklarativt program i moduler. Programmoduler kan utvecklas separat och sedan kombineras på olika sätt, vilket sparar tid för programmerarna i det långa loppet. En deklarativ statisk analys kan ganska enkelt användas som en modul inuti en större, mer komplicerad, analys.

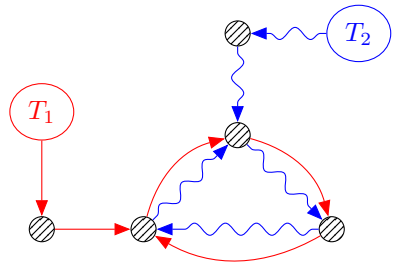
En statisk analys kan delas upp i små deklarativa delar som kallas *attribut*. Attributen kan enkelt kombineras till moduler som sedan används för att utveckla nya analyser.

Ett sätt att tänka på modulerna i en statisk analys är som pusselbitar. En pusselbit (modul) kan läggas till i ett pussel (en statisk analys) så länge den har rätt form (programgränssnitt):



Ett viktigt forskningsresultat i den här avhandlingen är en ny algoritm för att kunna beräkna flera attribut samtidigt. Detta kan förkorta beräkningstiden: i ett experiment visade mina mätningar att kompileringstiden kunde kortas med hälften. En annan fördel är att svarstiden kan minskas i interaktiva programmeringsverktyg: med samtidiga beräkningar måste man inte vänta på att föregående beräkningar är klara innan man börjar på den nästa. Mina experimentella resultat visar att svarstiden kan minskas från sekunder till under en millisekund.

För att kunna utföra beräkningar samtidigt har de flesta datorer idag flera *kärnor*, där varje kärna exekverar varsin *beräkningstråd* jämnlöpande med andra kärnor i datorn. Med min nya algoritm kan den sammanlagda beräkningstiden för attribut minskas genom att dela upp beräkningsarbetet på flera (beräknings)trådar. När en tråd beräknat ett attribut sparas värdet och kan direkt användas av en efterföljande tråd som då slipper göra om beräkningsarbetet. Uppdelning av arbetet för fem attribut med två jämnlöpande trådar illustreras i figuren nedan.



Figur: Illustration av två trådar, T_1 och T_2, som samtidigt beräknar fem attribut (randiga cirklar). Beräkningsflödet för T_1 visas med röda pilar, och för T_2 med vågiga blåa pilar. Beräkningen är i det här fallet *iterativ*, och går runt i en cirkel tills det rätta värdet har beräknats. Med min algoritm kan varje tråd utföra sin beräkning i varje attribut utan att behöva vänta på den andra tråden. Dessutom hjälper trådarna varandra genom att en tråd använder resultatet från den andra tråden om den andra hann före.

Kompilatorer

Statiska analyser är centrala i kompilatorer, det vill säga program som översätter programmerarens kod till maskinkod som en dator kan köra. I mitt arbete har jag arbetat främst med en kompilator som heter ExtendJ, som kompilerar program skrivna i programmeringsspråket Java. ExtendJ är fullt deklarativt programmerad och använder attribut, vilket gör det förhållandevis enkelt att använda ExtendJ för att bygga nya statiska analyser för Java.

För min avhandling har jag dels utvecklat nya analyser som bygger på ExtendJ, dels förbättrat ExtendJ självt.

Bland de statiska analyser som jag utvecklat i ExtendJ finns en analys som kan användas för att minska testningstiden för Java-program. Idén är att endast köra om de test som kan påverkas av den senaste ändringen i programmet som testas. Med denna teknik lyckades jag halvera den totala tiden för testning av flera olika program.

En av förbättringarna jag gjort i ExtendJ var att uppdatera kompilatorn till att stödja en ny version av Java. Sammanlagt har förbättringarna som jag gjort i ExtendJ gjort det enklare för forskargrupper runt hela världen att bygga nya statiska analyser och språktillägg till Java.