



Professor rekommenderar försiktighet

När AI skapar mjukvara

När Robert Feldt var liten drömde han om datorer som skrev sina egna program.

Nu hjälper han drömmen att bli verklighet men varnar samtidigt för riskerna.

Du skriver ett sms och mobilen gissar hela tiden nästa ord. Känner du igen det? Och ditt mailprogram ger förslag på korta brevvar.

Det kallas autokomplettering och finns numera även som verktyg för den som skriver källkod. Den väntas bli allt smartare. Idag kompletterar den enstaka ord. Snart kanske den producerar hela kodavsnitt som förslag på ett enda ord, efter att ha gissat vad du försöker göra.

AUTOKOMPLETTERING är ett av flera exempel på smarta verktyg som finns eller snart kommer att finnas i en utvecklingsmiljö nära dig. Andra smarta verktyg söker efter buggar, väljer testfall eller förenklar, optimerar och till och med korrigerar kod. Eller ger dig ett röstgränssnitt till din utvecklingsmiljö.

Programmerare har alltid varit experter på att ersätta sig själva med automater och verktyg. AI

öppnar nu ytterligare möjligheter.

Vad gäller smart komplettering finns den idag för bland annat Java, Python, Kotlin och Dart. Verktygen har namn som Codota, Kite och ML Complete.

– De är idag mer eller mindre bra. De kommer att fortsätta att utvecklas, och bli vardag, men de kommer aldrig att bli perfekta, för de kan inte veta vad du vill uppnå.

Den bedömningen kommer

från Robert Feldt, professor i mjukvaruutveckling på Chalmers. Han deltar i utvecklingen av denna typ av automatisering, och håller ett öga på den. Även ett kritiskt öga. Trots att det här faktiskt är vad han drömde om när han var liten.

– Tänk att bara kunna ligga i sin säng och prata med datorn för att instruera den! Den ambitionen har jag hållit kvar i bakhuvudet.

Han sålde sitt första program

när han var tretton och doktorerade på Chalmers år 1997 för Jan Thorin som gav honom fria händer – ”jag vet inget om mjukvara utom att det är framtiden, och jag litar på dig”.

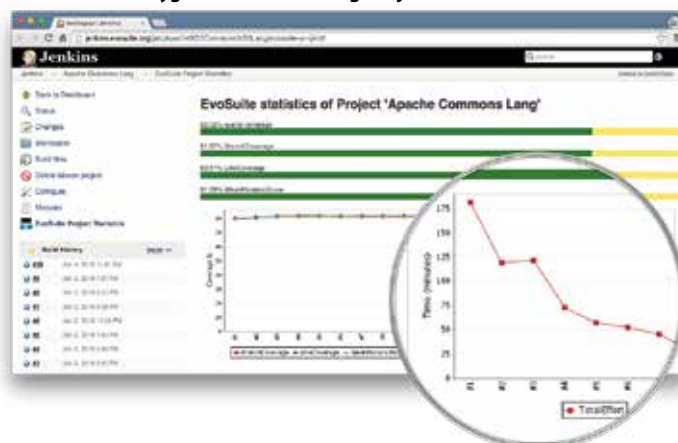
DET HETA OMRÅDET djup maskinlärning bidrar till nyutveckling inom utvecklingsverktyg. Det gör även klassisk AI och andra datalogiområden, separat eller i korsbefrukning med varandra.

– Deep Learning är den nuvarande flugan och den har inneburit en stor renässans för området som helhet med både ökat intresse och nya pengar in.

– Men i många tillämpningar antingen behövs inte Deep Learning för att enklare metoder presterar lika bra, eller så är den något bättre prestandan inte värd den mer komplexa träning och de mer svårförståeliga resultat som Deep Learning ofta kan leda till.

Automater hänger inte bara över axeln på dig när du

Smarta testverktyg hittar in i utvecklingsmiljöerna.



kodar. De kan också lägga sig i resultatet, som att skapa testfall, optimera kod eller rätta buggar. Idag handlar det om små ändringar, men förmodligen blir de allt större med tiden.

En spännande optimeringsmetod är att helt sonika radera kod för att göra systemet snabbare eller strömsnålare. Det kanske till och med introducerar fel, men kanske bara för vissa specialfall som koden inte behöver kunna hantera? Eller så sänker det noggrannheten vilket kan vara ett pris värt att betala för högre prestanda.

I ARTIFICIELLA NEURONNÄT går det ibland att minska antalet precisionsbitar från 16 till 8, eller skära bort grenar.

– Forskning pekar på att de stora komplexa modellerna sällan behövs, speciellt inom känsliga tillämpningar, som inbyggda system.

Robert Feldt vill inte beskriva de nya AI-verktygen som en revolution. Det är inte så att robotar håller på att ta över programmerarnas jobb.

– Smarta verktyg kommer att ta över mer och mer av utvecklingsarbetet. Gränserna kommer att förskjutats till och mer intressanta uppgifter. Men vi fortsätter använda människor för sådan som är kreativt och svårt.

– Det kommer inte att komma helt nya sorters verktyg, utan snarare kommer alla verktyg att bli något lite bättre, med hjälp av AI och maskininläring.

Hans egen forskarkarriär har handlat mycket om automatisk programtestning. Det började redan vid examensarbetet. Han skulle optimera flygteknik med hjälp av en AI-metod kallad genetiska algoritmer.

Algoritmen hittade till hans överraskning omedelbart en optimal lösning. Men det visade sig att programmet tjuvåkade på en bugg i flygsimulatorens kod. Idag finns gott om exempel på AI som "fuskat" på liknande vis.

ROBERT FELDT INSÅG att det gick att använda genetiska algoritmer och andra sökalgoritmer för att hitta buggar. Han blev därefter en av pionjärerna inom området, som generellt kallas sökbaserad kodutveckling.

I ett samarbete med sydkoreanska forskare har Robert Feldt

utvecklat ett sätt att testa neuronät i autonoma fordon.

– Grundidén är att ägna det mesta av testandet åt saker som är lagom konstiga – som en regnbågsfärgad bil. Och mycket mindre test för osannolika saker, som ett flygplan som kommer in från vänster och en människa med get från höger.

Facebook köpte för ett par år sedan ett centralt startupföretag inom sökbaserad testning och använder dess verktyg Sapienz på alla sina appar – och har tidvis hittat hundratals buggar i månaden. Ett annat verktyg kallat Sapfix föreslår därefter en buggfix.

Det finns också ett öppen-kodsverktyg, Evosuite, som gör sökbaserad testning av Javakod.

Om du som läsare börjar känna dig oroad över att allt mer kodande överläts åt maskiner – så är du i gott sällskap. Den oron är faktiskt en av Robert Feldts specialiteter – att få företag att förstå riskerna med att introducera AI-verktyg.

Robert Feldt förespråkar att AI introduceras på en så låg nivå som möjligt – där konsekvenserna av misstag är som minst.

En AI som bara hjälper dig hitta testfall innebär en låg risk. En AI som rättar kod påverkar produkten och riskerna är högre – sådana rättelser bör granskas.

DEN HÖGSTA RISKEN är ett AI-system som uppdaterar sig medan det körs. Det är ovanligt i säkerhetskritiska tillämpningar, men Tesla och andra tycks driva utvecklingen i den riktningen.

Robert Feldt har sett flera exempel i företag där distinktionen mellan de här nivåerna inte görs när ledningen bestämmer att AI ska introduceras i verksamheten.

Många AI-baserade utvecklingsverktyg finns fortfarande bara i forskarnas labb.

– Själva forskningen om AI-verktyg är mogen och visar ganska övergripande på stora vinster. Dels i att effektivisera existerande testning, men även

FAKTA

Verktyg på gränsen till genombrott

Elektroniktidningen vill veta vad som är hetast just nu inom AI-stödd mjukvaruutveckling. Robert Feldt väljer tre områden: bollar, fuzzing och kausalanalys:

• **Bollar** är utvecklingsmiljöer som det går att prata med på naturligt språk, istället för att ge textkommandon. Nyttan är helt enkelt att de borde kunna spara tid.

– Det är ett område som det varit mycket tryck på, på sistone. Ingen orkar minnas långa Git-kommandon.

Git är en populär kollaborativ utvecklingsmiljö.

– Hellre vill man kunna bara prata, och säga till exempel "visa vilka andra som pillat i modulerna som det verkar vara fel på".

Framstegen inom djup maskininläring ligger bakom bollarnas utveckling. Det är den som används för att tolka naturligt språk.

– Men det känns fortfarande lite grand som att det är mer explorativ forskning som görs och vi har ännu inte sett utvärderingarna som visar vilka vinsterna verkligen är på större delar av det som utvecklare gör dagligen.

• **Fuzztest** är automatiska sätt att välja testdata till program. Det är teoretiskt omöjligt att testa alla tänkbara indata. Att generera helt slumpmässiga indata blir ineffektivt. Fuzztest är att generera underliga indata som ser naturliga ut men ändå provocerar fram oupptäckta fel.

Här finns en del mogen teknik. Fuzztest har hunnit bygga upp en lång meritlista över identifierade buggar i populära program. Facebook hittar tusentals buggar i mobila appar genom att testa på det viset. Programmet American fuzzy lop är öppen källkod och har hittat mängder av buggar i öppen källkod.

Särskilt intressant är fuzztest för säkerhetshål. De beror ofta på oväntade indata som får program att sänka garden och släppa igenom inkräktare. Flera omtalade håll och hemligheter har hittats eller skulle kunna ha hittats via fuzztest.

• **Kausalanalys** (root cause analysis) handlar om att hitta grundorsaken till programfel. Felet finns ofta inte där det uppträder, utan kan behöva nystas upp. Det finns tecken på att maskininläring kan komma att göra stor skillnad inom området. En forskningsartikel i sommar kommer att rapportera ett test där 37 utvecklare kunde identifiera rotorsaken till 96 procent av undersökta programfel.

– Står vi inför ett genombrott nu? Det är för tidigt för att det ska kunna komma verktyg. Men det är en kittlande inspiration.

– På fem års sikt tror jag definitivt att det finns stöd för kausalanalys.

för att ta fram nya tester och automatiskt testa bättre än vad människor kan göra.

Men där tar utvecklingen ofta stopp. Verktyg som tas fram inom forskningen har ofta problem att ta sig fram till industrin. Det finns spännande forskningsresultat inom området som ingen tagit sig för att integrera i utvecklingsmiljöer.

– Forskare får sällan pengar till att göra nyckelfärdiga verktyg. Så det finns glapp mellan vad som är möjligt att göra, och vad som kan laddas hem och testas.

Han exemplifierar med en metod för property based testing (PBT) som utvecklades på Chalmers för 20 år sedan. PBT

testar att angivna samband gäller mellan in- och utdata genom att generera testfall.

– Det är en erkänt bra teknik. Forskare har haft verktyg. Men det är först nu som det dykt upp en rimligt robust version i öppen källkod, Hypothesis.

ÄVEN NÄR SAMARBETEN med industrin resulterar i en färdig lösning, är det sällan en generell lösning, och sällan det formas en avknoppare som skulle kunna sälja och underhålla ett verktyg.

Finns det något sätt för ett industriföretag kunna dra nytta av de nya resultaten? Robert Feldt ger ett anspråkslöst förslag till en ingång – exjobb.

– Om du är intresserad – hitta en teknik som är relativt mogen och låt en student göra ett jobb hos dig. Genom att ha exjobbare igång och testa och tillämpa nya tekniker i sin verksamhet, lär man sig mer och utvecklar sin arsenal.

JAN TÄNGRING
jan@etn.se

```
private void demo() throws Exception {
    URL url = new URL("https://www.codota.com");
    HttpURLConnection conn =
        (HttpURLConnection) url.openConnection()
        Mockito.mock(HttpURLConnection.class)
        Super completions by Codota.js
}
Codota ger förslag.
```