Programmeringsforskning i skolen - før og nå

Av Kristina Litherland

Å lære elever å programmere på skolen er ikke en så ny idé som noen tror. Faktisk befinner vi oss nå i det forskere gjerne kaller *den andre bølgen* av programmering i skolen. Formålet med dette kapitlet er at du skal kjenne til de viktigste trekkene ved bølgene av programmering i skolen, hvordan disse har blitt utforsket forskningsmessig og hvordan det er relatert til tilnærmingen i norsk skole i dag. Jeg starter med en presentasjon av de to bølgene før jeg går over til en mer generell diskusjon av forskningen.

Bølge 1: Papert og konstruksjonismen

Den første bølgen av programmering i skolen startet allerede på 1960-tallet i USA. Mannen som ofte får æren av å stå bak, er Seymore Papert. Papert (1980) sine ideer bygget blant annet på en motstand mot at elever ble «programmert» av spesifikt designet programvare spesielt, eller av lærere og læreplanverk generelt, som skulle forme dem til å bli gode i matematikk eller andre ting. Han snudde det hele på hodet. I stedet for at barna var passive brukere av et program eller mottakere av informasjon, skulle de være aktive *skapere* av programmer og det han kalte «mikroverdener» som inneholdt objekter elevene skulle «tenke med». Med dette ble Papert *konstruksjonismens* far, som er et perspektiv på læring gjennom å skape (fysiske og digitale) objekter. Gjennom å lage programmer i det tekstbaserte programmeringsspråket LOGO skulle elevene ikke bare lære matematikk, men også lære å *tenke*.

Selv om den første bølgen av programmering i skolen hadde gode intensjoner, ble den aldri så stor som forkjemperne håpet. Det skyldtes blant annet lav tilgang til datamaskiner rundt om i verdens klasserom, at konstruksjonismen ikke lot seg integrere med andre mer etablerte perspektiver på læring i skolen, men også at resultatene ikke stod til forventningene. Da forskere utover på 80-tallet begynte å dykke ned i Paperts ideer og metoder klarte de ikke å finne tydelig evidens for at elevene faktisk ble flinkere i matematikk eller lærte å tenke bedre (Lye & Koh, 2014). Med det ble programmering i skolen-prosjektet stort sett lagt på is innen 1990. Forskningen på programmeringspedagogikk fortsatte, men nesten utelukkende i høyere utdanning, og uten konstruksjonisme som utgangspunkt. I tillegg fortsatte noen skoler til en viss grad å undervise tradisjonell programmering som valgfag, uten de store ideene om programmering som en slags generell tilnærming til læring for alle. Det skulle ta flere tiår før noen igjen tok opp tråden.

Bølge 2: Wing og computational thinking

Jeanette Wing er en amerikansk forsker som ofte trekkes frem som en av grunnleggerne av den andre bølgen av programmering i skolen. I 2006 skrev hun et innlegg (Wing, 2006) til et informatikk-tidsskrift der hun erklærte at «alle» burde lære seg den grunnleggende ferdigheten *computational thinking* (CT, direkte oversatt: komputasjonell tenking). På norsk kjenner vi CT som algoritmisk

tenking, en oversettelse som ikke nødvendigvis fanger opp bredden av hva CT kan være. Å komputere betyr å utregne eller å beregne, og historisk ble de som kunne beregne datoen for fremtidige påsker kalt komputister (computers). I dagens samfunn opplever mange at datamaskiner nærmest har tatt beslag på det å gjøre beregninger, selv om det er noe alle mennesker gjør hver eneste dag. Det er lett å glemme at det på en eller annen måte er ett eller flere mennesker som står bak enhver utregning gjort av en datamaskin. Det er kompetansen til disse menneskene CT prøver å fange opp. CT handler derfor ikke om tenking, selv om navnet tilsier det, men om praktisk utvikling av løsninger på problemer.

Diskusjon

Før vi går over til diskusjonen er det viktig å merke seg at selv om jeg har presentert de to bølgene, så har det – som nevnt – vært mer eller mindre kontinuerlige initiativer rundt omkring i verden til å lære elever å programmere i skolen gjennom valgfag. Både på ungdomsskolenivå de siste årene, og i mange år i videregående skole, har norske elever hatt mulighet til å velge programmering/informatikk som valgfag. Jeg kaller dette for den tradisjonelle tilnærmingen til programmering i skolen. Forskjellen på denne og de to bølgene er at bølgene representerer tilnærminger til programmering for *alle* elever, ikke bare dem som velger det som valgfag. I tillegg er målene annerledes. I spesifikke programmeringskurs er målet typisk at elevene skal lære programmeringsfaget mer profesjonsrettet. I de to bølgene kan man argumentere for at programmeringsfaglig kunnskap ikke er hovedmålet. Felles for de to bølgene er nemlig at de begge er bygget på store idéer om hvilke positive effekter programmeringskompetanse kan ha, annet enn å kunne ha programmering som jobb.

På samme måte som Paperts ideer ble lagt under lupen, utforskes Wings ideer om CT empirisk i disse dager. Innlegget hennes tillegges mye vekt i programmeringsforskningen og -retorikken, men det er viktig å merke seg at det ikke er empirisk begrunnet. Innlegget har faktisk ikke en eneste kilde, og det er ingen bred enighet om hvordan CT faktisk skal defineres.

En utfordring ved CT-perspektivet er at det innebærer en tilnærming til programmering som en generell kompetanse som er forhøyet over bestemte programmeringsspråk. Spørsmål som diskuteres i CT-litteraturen er om og hvordan det er mulig å lære CT uten å lære bestemte programmeringsspråk, og om elever som har lært et første programmeringsspråk klarer å identifisere de generelle konseptene og praksisene som brukes i dette ene språket og overføre dem til andre språk og kontekster. Hvilken verdi har egentlig programmeringskompetanse uten noen kontekst? Dette er spørsmål som ikke har vært like relevante tidligere, særlig ikke i den mer profesjonsrettede tilnærmingen til programmering der målet typisk er å mestre bestemte språk. En kjent forsker på feltet, Mark Guzdial, har uttalt at det er vanskeligere å lære sitt andre programmeringsspråk enn sitt første, fordi ulike språk har ulike regler som kan være motstridende, og forståelsen du fikk av det første språket kan kanskje ikke overføres til det andre. Dersom dette stemmer kan det bety at programmeringsundervisning ikke nødvendigvis bidrar til en generell forståelse av hva programmering er og hva det kan brukes til, eller at denne

generelle forståelsen helt enkelt ikke eksisterer. Altså at programmeringskunnskap ikke kan eksistere utenfor en bestemt kontekst, og at CT-idéen forblir nettopp en idé. Det kan også bety at CT må undervises på bestemte måter som skiller seg fra hvordan vi typisk har undervist i programmering. Alt dette er sentrale punkter i CT-debatten, som kommer til uttrykk gjennom blant annet initiativer til såkalt analog programmering.

I en litteraturgjennomgang fra 2014 prøvde forskerne Lye og Koh å få oversikt over CT-feltet. De sorterte 27 relevante studier basert på om de handlet om de typiske programmeringskonseptene, om de handlet om praktisk bruk av disse, eller om de handlet om perspektiver på CT og refleksjon rundt hvordan CT er del av livene våre. Av 27 studier, handlet 23 om konsepter, 6 handlet om praksiser, og kun 2 tok for seg refleksjoner. Noen studier dekket flere områder. Dette forteller oss at selv om mange mener vi fra 2006 har vært i en bølge som tar for seg en generell og praktisk tilnærming til problemløsning med datamaskiner, så er forskningen fortsatt sterkt rettet inn mot forståelse av de tradisjonelle programmeringskonseptene.

En viktig debatt i samtidsforskningen handler om skillet mellom blokk-basert og tekst-basert kode. I den første bølgen fantes ikke blokk-basert kode, og mange forskere mener blokk-basert kode er noe av grunnen til at det er verdt å gi «programmering for alle»-prosjektet et nytt forsøk. Begge tilnærminger har fordeler og ulemper, og forskningsmessig kan vi ikke si at det ene er bedre enn det andre, men det henger sammen med målet med undervisningen. Det er vanskelig å se for seg at profesjonelle utviklere ikke skal ha erfaring med tekst-basert kode, men dersom målet er mer generelt så er det ikke nødvendigvis slik at blokk-basert kode fungerer dårligere. Som med mye annet er dette også et felt som utvikles, og i årene som kommer vil vi forhåpentligvis få en bedre forståelse av likheter og ulikheter i læringen som skjer når man bruker blokker eller skriver tekst.

Et annet skille mellom programmering før og nå som ikke er like tydelig kommunisert, er at moderne blokk-baserte språk typisk er lagt opp til å være hendelsesbaserte. Det betyr at elevene kan lage programmer der det er enkelt å legge til rette for brukerinteraksjoner gjennom å trykke på knapper eller bruk av ulike sensorer. Hendelsesbasert programmering har vist seg å appellere godt til nybegynnere (Grover, 2020), og gjør veien kortere til å utvikle løsninger som kan brukes i «den virkelige verden» i stedet for tradisjonelle tekstbaserte språk som i sin reneste form tar for seg utregninger på en skjerm.

Den norske modellen, der programmering er integrert i andre fag, kan ses som en kombinasjon av de to bølgene av programmering i skolen. Papert brukte programmering i matematikkfaget, men perspektivet var litt annerledes enn det vi finner i læreplanen. Han mente at man lærer «å tenke matematikk» gjennom å skape programmer, mens i læreplanen er programmering i større grad tenkt som en metode man kan bruke for å løse ulike faglige problemer. CT er representert gjennom problemløsning. Programmering som metode i fag er et relativt uutforsket felt, og vi kan ikke med

sikkerhet si hva det vil medføre for lærere eller elever. Læreplanen legger spesielt vekt på at elevene skal bruke de grunnleggende programmeringskonseptene (variabler, vilkår og løkker), som kan tolkes som at den tradisjonelle programmeringspedagogikken ligger til grunn. Dette gjør at det norske prosjektet med programmering i skolen for alle er veldig komplekst. Elevene skal både lære konkrete programmeringsspråk de kan bruke i praksis, de skal lære faget de jobber med, og de skal lære en slags generell problemløsningstilnærming (CT) som er vanskelig å definere og som er uavhengig av spesifikke fag, inkludert programmeringsfaget.

Hva nå?

Selv om mange føler at programmering i skolen representerer noe nytt (og litt skremmende?), så har vi nå sett at dette egentlig er et felt som har vært i utvikling gjennom flere tiår. I dag har vi mange flere verktøy tilgjengelig for å støtte læringen enn vi hadde tidligere, både rent teknisk, men også på et mer teoretisk plan. Programmering kan være så mye mer i dag enn det har vært tidligere.

Refleksjonsspørsmål

Hva mener du kan være styrker og svakheter ved de tre ulike tilnærmingene (tradisjonell programmeringsundervisning, konstruktivisme (bølge 1) og CT (bølge 2))?

På hvilken måte kan de ulike tilnærmingene påvirke hvordan man legger opp undervisningen?

Hvordan er tilnærmingen i læreplanen knyttet til den historiske utviklingen av programmering i skolen?

Referanser

Grover, S. (2020). *Computer Science in K-12: An A-To-Z Handbook on Teaching Programming*. Edfinity.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51-61.

Papert, S. A. (1980). *Mindstorms: Children, computers, and powerful ideas*. Basic books.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, *49*(3), 33-35.