Ferdighetsbasert programmeringsopplæring

Av Kristina Torine Litherland

Hvordan kan vi legge opp programmeringsopplæring? Hvilke ferdigheter er det elevene egentlig trenger å opparbeide seg for å kunne bruke programmering i fag? Tradisjonelt sett har programmeringsopplæring vært preget av et tydelig forelesningsformat der elevene (eller rettere sagt: studentene) kopierer kode (programtekst) foreleseren forklarer og skriver på tavla. Det er i hovedsak to grunner til dette. Den første er at programmering først og fremst har blitt undervist i høyere utdanning der forelesninger har vært den ledende opplæringsformen «i alle år». Da er det ikke så rart at også programmeringsundervisningen adopterer dette formatet. Den andre grunnen er at programmeringsspråk historisk sett har hatt en streng og komplisert grammatikk som studentene måtte mestre før de ville få til å produsere noe som helst.

De siste 10-20 årene har det blitt utviklet flere blokkbaserte programmeringsspråk. Fordelen med disse er at den kompliserte grammatikken i stor grad er innebygget i grensesnittet, som betyr at utvikleren ikke trenger å kjenne til reglene for å produsere noe meningsfullt. Som følge av denne utviklingen har terskelen for å programmere blitt mye lavere, som gjør at yngre aldergrupper (barn i skolealder) enklere kan komme i gang med programmering. Blokkbasert programmering har også skapt en motreaksjon til det strenge forelesningsformatet, der programmeringsopplæringen blir sett på som en åpen og kreativ prosess der de lærende fritt skal bestemme hva de ønsker å utvikle, og at instruksjon kun skal foregå når disse selv etterspør det for å fullføre egne prosjekter (Resnick, 2017).

Retter vi et kritisk blikk mot disse to metodene (forelesning og frie aktiviteter) er det problematiske sider ved begge, særlig i forbindelse med at programmering skal inn i norsk skole. Felles er at de behandler programmering og utvikling av programmer som målet. I fagfornyelsen er programmering integrert i andre fag, og ikke et fag i seg selv. Vi kan derfor ikke kun ha fokus på hvordan man teknisk produserer programmerer, men hvordan programmering kan brukes i fagene. I tillegg handler programmering og programmeringsforståelse om mer enn bare å kunne kopiere og produsere (se nedenfor). Videre er ikke nødvendigvis forelesning som metode i seg selv problematisk. Det problematiske er når denne metoden blir enerådende og brukes selv når den kanskje ikke er nødvendig. Blokkbasert kode har tross alt tonet ned behovet for mye instruksjon før man begynner å programmere på egenhånd. På den andre siden er det ikke slik at helt åpne prosjekter kommer uten utfordringer, og heller ikke denne tilnærmingen burde være enerådende i klasserommet. Det er jo ikke alltid frie prosesser er så frie som vi tror og håper, og frihet i seg selv lar seg ikke alltid kombinere med læreplaner og -mål.

Alternative aktiviteter

Disse to paradigmene i programmeringsopplæring er så utbredte at det kan være vanskelig å se for seg andre måter å legge opp undervisningen på. En fremgangsmåte er å vurdere hvilke ferdigheter som er involvert i gode programmeringsprosesser, og så jobbe spesifikt med disse. Det er fort gjort å tenke at programmering først og fremst handler om produsering, men det er egentlig bare toppen av ferdighetskransekaka.

Å lese kode

Den kanskje mest grunnleggende ferdigheten generelt i skolen, og kanskje mest ignorerte i programmeringssammenheng, er lesing. Visste du at forskere har funnet en sammenheng mellom programmeringsforståelse og å kunne lese kode høyt? (Hermans et al., 2018) Det er kanskje ikke så overraskende. Det som er overraskende er at vi ikke har noen tradisjon for å jobbe spesifikt med

lesing i programmeringsopplæring. Kan du ikke lese kode er det vanskelig å se for seg at du kan forstå eller produsere den. Derfor burde lærere legge til rette for aktiviteter der elever leser kode, både individuelt og høyt for andre, inkludert kode de ikke har sett før. Sistnevnte innebærer at elevene får mulighet til å predikere.

Å predikere

Å predikere handler om å se for seg hva resultatet kan bli når et program kjøres, uten å kjøre det først. Det krever altså at man leser koden og prøver å forstå hva programmet skal gjøre. Denne ferdigheten er ikke til stede i forelesningsmodellen der all kode blir forklart. Ofte er den heller ikke til stede i helt frie aktiviteter, siden mange elever bruker en «prøve og feile»-strategi som ikke innebærer at de reflekterer over programmets funksjon før de tester det. Programmeringsopplæring burde derfor inneholde eksplisitte predikeringsoppgaver. Det viktigste er ikke predikasjonene i seg selv, men at de skaper en mulighet for å diskutere koden. Predikeringsoppgaver krever altså at man skaper en kultur i klasserommet som ikke handler om å gi riktige svar, men om å komme med forslag og teorier som kan diskuteres. En avkrefta hypotese er nemlig ikke en dårlig hypotese.

Å snakke om programmering

Å snakke om og forklare programmer er en viktig ferdighet, for eksempel om man skal predikere hva et program gjør. Programmering foregår ofte i samarbeid med andre, og derfor er det viktig å kunne diskutere og reflektere rundt hva programmene betyr, både fra tekniske og faglige perspektiver, men også for oss som mennesker eller samfunnet generelt. Er det alltid læreren som står for forklaringene, får ikke elevene anledning til å trene på denne ferdigheten. Programmering handler ikke bare om å lage programmer som forstås av datamaskiner; de skal også forstås av mennesker. Derfor er det viktig at elevene får ferdigheter i å uttrykke seg om dem muntlig, men også skriftlig gjennom å kommentere programmer.

Å endre og gjenbruke programmer

Vi skiller mellom gjenbruk av kode (endre på kode slik at den kan brukes i nye kontekster) og det å kopiere kode som en egen aktivitet (at eleven skriver av lærerens kode linje for linje eller blokk for blokk). Selv om å kopiere kode er en av de mest utbredte metodene innen programmeringsopplæring, og det er vanlig praksis å gjenbruke kode i programmeringsindustrien, argumenteres det for kodekopiering ikke gir stort læringsutbytte fordi det å kopiere kode nøyaktig er både vanskelig og tar mye tid. Det blir ofte mange feil og mangler i kode som kopieres av nybegynnere. I stedet mener forskere at man heller kan gi elevene ferdig, fungerende kode og be dem endre på den, altså å gjenbruke kode. Da unngår man at elevene sitter med et dårlig/ikke-fungerende kopiert program som utgangspunkt, og de kan heller bruke tiden på å gjenbruke programmet de har fått utdelt til å skape noe nytt.

Å produsere programmer

Den mest avanserte ferdigheten er selvfølgelig det å produsere egne programmer. Det krever å kunne lese, predikere, forklare og endre på eksisterende programmer. Dersom elevene ikke har disse ferdighetene i bunn, kan produseringsprosessen ofte bli en kopieringsseanse eller at de legger til, fjerner, flytter og endrer på blokker/kode uten å forstå hva de egentlig driver med. Kopiering og «fri» koding kan fungere dersom målet er å lage avanserte programmer uten nødvendigvis trenge å forstå dem, eller at elever i interessebaserte miljøer skal få utfolde seg kreativt, men i en kontekst der programmering skal brukes som en systematisk problemløsningsprosess i fag kan vi ikke kun basere undervisningen på kodekopiering eller fullstendig åpne prosjekter.

Nå har vi gått gjennom noen ulike aktiviteter du kan bruke i klasserommet for å trene spesifikke programmeringsferdigheter. Som du ser må ikke elevene nødvendigvis produsere så mye kode for å lære programmering, i denne modellen er det siste steg. Å legge opp programmeringsundervisning på måten presentert her er utviklet av et forskerteam i Storbritannia, og metoden kalles PRIMM (Predict, Run, Investigate, Modify, Make). Ønsker du å lære mer kan du lese Sentance et al. (2019).

Refleksjonsspørsmål

I programmeringsundervisning skiller vi ofte mellom digitale og analoge aktiviteter. Hvordan vil du kategorisere disse aktivitetene?

Hvordan kan metoder du kjenner fra andre fag brukes i programmeringsopplæring?

Hva er gode refleksjonsspørsmål å stille elevene når de jobber med programmering?

Referanser

Hermans, F., Swidan, A., & Aivaloglou, E. (2018, May). Code Phonology: an exploration into the vocalization of code. In Proceedings of the 26th Conference on Program Comprehension (pp. 308-311).

Sentance, S., Waite, J., & Kallia, M. (2019). Teaching computer programming with PRIMM: a sociocultural perspective. *Computer Science Education*, 29(2-3), 136-176.

Resnick, M. (2017). *Lifelong kindergarten: Cultivating creativity through projects, passion, peers, and play*. MIT press.