Jan Arild Dolonen, Anders Kluge, Kristina Litherland og Anders Mørch
Universitetet i Oslo

Litteraturgjennomgang av programmering i skolen

08.11.2019

UiO : Institutt for pedagogikk
Det utdanningsvitenskapelige fakultet

Sammendrag

Som en del av fagfornyelsen skal programmering innføres i flere obligatoriske fag fra høsten 2020. Formålet med denne rapporten er en litteraturgjennomgang av hva forskning på programmering og læring viser, og å undersøke det utdanningspolitiske grunnlaget for å innføre programmering i norsk skole. Rapporten er en del av forprosjektet "Programmering i (ungdoms-)skolen", og er et samarbeid mellom KompAks AS og Institutt for Pedagogikk ved Universitetet i Oslo, finansiert gjennom FORREGION Oslo Akershus/Oslo Edtech Cluster og Norges Forskningsråd (prosjekt# 304162).

Argumentene for å innføre programmering i skolen er primært at programmering kan bidra med nødvendige digitale ferdigheter i befolkningen, er en naturlig del av problemløsning i ingeniørfagene, og overlapper med kritisk tenkning i fagfornyelsen. Forskningsgrunnlaget virker imidlertid tynt. Tidligere forskning tyder på at ferdigheter i programmering i liten grad lar seg overføre til andre disipliner og problemstillinger, men samtidig erkjenner man at digitaliseringen av samfunnet krever utvidet kompetanse. Det finnes også lite robust forskning på adekvate undervisningsopplegg. To lovende opplegg er imidlertid FACT-metoden som fokuserer på balansen mellom strukturerte og ustrukturerte læringsaktiviteter, og PRIMM-modellen som fokuserer på samarbeid og muntlighet i kombinasjon med prediksjon, utprøvning, og refleksjon. Sistnevnte type modell har vist seg å skape dyp og varig forståelse i andre fag.

Tilrettelegging for programmering i skolen skjer gjennom ulike tiltak i regi av Utdanningsdirektoratet (Udir). De stiller midler til rådighet gjennom “den teknologiske skolesekken”, og har i tillegg lansert en modell for lokal kompetanseutvikling i skolen i samarbeid med UH-sektoren. Hvilken rolle det private næringsliv har i denne satsingen er uklar, men det virker som de må operere som tidligere gjennom støtteordninger i regi av Udir eller forskningsrådet. Programmering innføres relativt raskt i skolen, og gir betydelige institusjonelle utfordringer med hensyn til lærernes kompetanse. Få lærere har kompetanse i å undervise i programmering, noe som gjør at det er et behov innen feltet for å finne frem til gode læringsressurser og undervisningsopplegg.

Innholdsfortegnelse:

1. Introduksjon **3**

2. Forskning på kognitive ferdigheter og overføringsverdi **5**

2.1. Programmering i skolen i dag 7

3. Forskning på undervisningsopplegg **10**

3.1. Undervisningsressurser 11

3.2. Programmeringspedagogikk 12

3.3. Mot mer forskningsbasert undervisning i programmering 12

3.4. Fra blokk til tekst 13

3.5. Fysisk programmering 14

3.6. Programmering for fag 14

4. Sentrale politiske aktørers argumenter for programmering i skolen **15**

5. Sentrale aktørers arbeid for å innføre programmering i skolen **20**

6. Oppsummering og diskusjon **24**

7. Referanser **28**

1. Introduksjon

Som en del av fagfornyelsen skal programmering innføres i flere obligatoriske fag i norsk skole fra høsten 2020 (NOU 2015:8, Meld. St. 28). I denne rapporten skal vi gjennomgå nasjonal og internasjonal forskning på programmering og læring, og selv om læreplanene ikke er endelig vedtatt skal vi undersøke det utdanningspolitiske grunnlaget for å innføre programmering i norsk skole. Formålet med denne litteraturgjennomgangen er å undersøke hva forskning på temaet fokuserer på, hva man har funnet frem til, og hvordan dette skal iverksettes i norsk skole. Forventet resultat er litteratur og funn som kan være av praktisk nytte for Kompaks i arbeidet med å videreutvikle sin tjeneste.

Det ble gjort betydelig forskning på programmering og læring på 1980-tallet først og fremst gjennom arbeidet til Seymour Papert[1] ved Massachusetts Institute of Technology (MIT). Imidlertid uteble de forventede resultatene, og forskningen forsvant delvis på 1990-tallet. Nå er "å lære gjennom programmering" tilbake som et hett tema i forskningen (Grover & Pea, 2013; Waite, 2018), og frontes også utdanningspolitisk gjennom fagfornyelsen (NOU 2015:8, Meld. St. 28) som er en fornyelse av alle læreplanene i norsk skole. I fagfornyelsen skal dagens fag i hovedsak beholdes, men struktur og innhold endres. Grunnen til fagfornyelsen er at mange av læreplanene har vært for omfattende med tanke på innhold. I fagfornyelsen vil man derimot legge mindre vekt på bredden av innhold og mer vekt på dybdelæring; det vil si gradvis utvikling og spesialisering av kunnskap og forståelse gjennom bruk og integrering av begreper, metoder og sammenhenger i spesifikke fag og mellom fagområder. Programmering har blitt frontet som en type aktivitet som kan bidra til dybdelæring gjennom sitt fokus på abstrahering, dekomponering og resonnering som gjør at elevene kan lære seg problemløsning og kritisk tenkning (Sevik et al., 2016).

Med dette som bakteppe vil vi i litteraturgjennomgangen av forskning på programmering i skolen fokusere på hvilke kognitive ferdigheter elever lærer gjennom programmering,

[1] http://www.papert.org/

om ferdighetene kan brukes tverrfaglig, og ulike typer av undervisningsopplegg som kan støtte utviklingen av elevenes programmeringsferdigheter. For å veilede oss gjennom dette arbeidet har vi utviklet følgende 2 forskningsspørsmål:

1. Hvilke kognitive ferdigheter er relatert til programmering, og kan ferdighetene la seg overføre til andre fag?
2. Hva karakteriserer undervisningsopplegg som er prøvd ut på ulike alderstrinn?

I litteraturgjennomgangen av det utdanningspolitiske grunnlaget vil vi fokusere på sentrale aktører og deres argumenter for å få programmering innført i skolen, og hva som gjøres for å tilrettelegge for dette. For å veilede oss gjennom dette arbeidet har vi utviklet følgende 2 forskningsspørsmål:

3. Hvem er sentrale aktører, og hvilke argumenter benyttes for å innføre programmering i skolen?

4. Hva gjør de sentrale aktørene for å innføre programmering i skolen?

Vi har brukt ulike metoder for å belyse disse spørsmålene. For å undersøke forskning på programmering og læring har vi brukt Oria[2] ved Universitetet i Oslo (UiO) der man kan søke opp trykte og elektroniske bøker, tidsskrifter, artikler, bokkapitler m.m. ved UiO og andre norske fag- og forskningsbibliotek. Disse søkene er også koplet opp mot internasjonale databaser slik at man får treff på all internasjonal forskning som UiO har tilgang til. I tillegg har vi brukt snøballmetoden der vi har tatt utgangspunkt i høyt siterte artikler om temaet, og lest sentrale artikler som de refererer til. Prosessen gjentas til man har tilstrekkelig med litteratur. For å undersøke utdanningspolitiske dokumenter har vi her vektlagt Googlesøk og snøballmetoden fordi slike dokumenter sjelden er å finne i fagbibliotek og databaser. I denne delen har prosessen gått ut på å finne offisielle dokumenter eller dokumenter skrevet av sentrale aktører, for deretter undersøke hvilke dokumenter disse refererer til, og som kan være relevante for dette prosjektet.

[2] https://uio.oria.no/

2. Forskning på kognitive ferdigheter og overføringsverdi

Den nåværende interessen for- og innføringen av programmering i skolen kan sees på som den "andre bølgen" rundt denne ideen. Den "første bølgen" kan tidfestes til starten på 1980-tallet og mye kan tilskrives Seymour Papert (Koschmann, 1997). Sammen med kolleger hadde han utviklet Logo som et spesialisert programmeringsspråk og satt det inn i en programmeringsomgivelse som var enkel å bruke. Med begrensede teknologi- og programmeringskunnskaper kunne elevene i denne omgivelsen få en virtuell skilpadde til å bevege seg i forskjellige retninger (Papert, 1980). Papert fikk unge elever til å oppnå imponerende resultater med disse verktøyene. Han brukte erfaringene fra disse forsøkene til å argumentere på den ene siden at det var mulig for elever i ung alder å programmere relativt avanserte algoritmer, men enda viktigere så hevdet han at den læringsprosessen en elev gjennomgikk ved å programmere og løse denne typen problemer burde være et ideal for kognitiv utvikling hos elever : "Powerful intellectual skills are developed in the process" (Papert, 1980, p. 60). Han hadde flere andre med seg på dette blant andre Nickerson (1983) som hevdet at: "computer programming is a vehicle for teaching thinking skills" og Bork (1981) som så programmering som: "... analytic thinking applicable to a broad class of problems".

Denne oppfatningen om programmering som generell kognitiv trening egnet for barn og unge bredte seg på 1980-tallet, særlig i USA, og ga også støtet til en argumentasjon om skolereform. Papert (1993) mente at programmering måtte gi støtet til en større reform av skolen fordi elevene faktisk mestret det, og at den ga generell problemløsningsevne og kognitiv trening. I en slik reform skulle programmering og programmeringskompetanse være en bærebjelke i det kommende faglige utviklingsarbeidet. Det er viktig også å se dette i relasjon til spredningen av teknologi som skjedde på begynnelsen av 1980-tallet. IBMs «personal computer» var blitt lansert som en datamaskin i 1981 som alle kunne ha hjemme, og bidro til at datateknologi ble avmystifisert og mer tilgjengelig.

Internasjonalt ble det utover 1980-tallet og tidlig 90-tall gjennomført flere undersøkelser av overføringsverdien av det å lære programmering. Ideen om programmering som generell kognitiv trening viste det seg vanskelig å finne støtte for (for en gjennomgang av denne forskningen, se Mayer, Dyck, & Vilberg (1986) og Pea og Kurland (1984)). Det var ikke mulig å påvise at de som lærte å programmere gjennomgikk noen kognitiv trening som var bedre enn annet skolearbeid. En redusert ambisjon for overføring av kompetanse fra programmering var å se programmering som en øvelse i problemløsning som kunne anvendes i andre situasjoner og fag. Kompetanse i å «løse problemer» er også et generelt felt, noe som det er vanskelig å avgrense og å definere. Dette gjør det problematisk å finne metoder for å påvise en sammenheng som går fra kompetanse i programmering til bedret generell problemløsningskompetanse. De studiene som har vært gjort er så godt som entydige på at en slik sammenheng ikke lar seg påvise, og konklusjonene heller også i retning av at en slik overføring ikke finnes som en direkte linje fra programmeringskompetanse til problemløsning på andre områder (Koschmann, 1997; Mayer et al., 1986; Salomon & Perkins, 1987). Også resultatene om hvorvidt programmeringskompetanse skulle gi fordeler i andre emner viste seg vanskelig å påvise (Soloway, Lochhead, & Clement, 1982).

Med denne utviklingen mistet den første bølgen for programmering i skolen mye av sin kraft. Man forlot både den generelle ideen om at programmering var en slags idealvei for kognitiv trening, og at det skulle være en metode for å lære generell problemløsning som skulle være nyttig i mange fag og i situasjoner. En kuriøs, men også interessant parallell med argumentene for programmering i skolen var å sammenligne dem med argumentene for latin som ble undervist som et grunnleggende emne for et par århundre siden. En viktig del av rasjonale dengang var at latin var tankens språk, og at med god kunnskap i latin så hadde elevene «grunnmønsteret» som gjorde all annen læring lettere (Koschmann, 1997; Mayer et al., 1986). En klassisk studie som ble gjort for å undersøke disse egenskapene som følge av å kunne latin ble gjort av Thorndyke (1923). Hans studie av såkalt overføring eller «transfer» har blitt en viktig referanse i den internasjonale litteraturen for å diskutere muligheten for at eksisterende kunnskap kan brukes i nye

sammenhenger. Thorndyke fant ikke at elever som ble utsatt for latinopplæring kunne lære seg matematikk eller naturfag lettere, eller at det hjalp det man da kalte deres «verbale eller non-verbale IQ». I følge Pea og Kurland (1984) inngår ideen i et mønster der symbolspråk sees på som samsvarende med et slags kognitiv grunnstruktur og der opplæring i denne grunnstrukturen vil gi en form for generelle kognitiv trening brukbare i andre fag og situasjoner. Matematikk, logikk og skrivestrukturer er andre slike ideer, som man ikke har funnet støtte for. Fra vår hjemlige debatt kjenner vi det også i argumentene for sjakk i skolen.[3]

Verden er en ganske annen i dag enn på 1980-tallet, og argumentene må studeres på nytt i lys av dagens situasjon. Vi har nå den digitale hverdagen innpå oss på en helt annen måte i arbeid skole og fritid, og betimelige spørsmål om fremtidens kompetanser blir stadig mer påtrengende. I sekkebetegnelsen "Kompetanser for det 21. århundre" ("21st century skills") er også digitale ferdigheter plassert, sammen med blant annet problemløsning, kritisk tenking, kreativitet, samarbeidsevner og kommunikasjonskompetanse (Brinkley et al., 2012). Den underliggende drivkraften i disse endringene er utviklingen av digital teknologi, og den økende tilgangen til - og spredningen av den. På den ene siden skal skolen gi opplæring i å håndtere teknologien (for eksempel med kritisk tilnærming til en ny medieverden), og på den annen side skal også elevene lære å utnytte mulighetene ved å bruke teknologien kreativt, til samarbeid, kommunikasjon og problemløsning.

2.1. Programmering i skolen i dag

Når vi skal studere programmering i skolen i dag er det viktig både å kjenne historien og anerkjenne at mye har endret seg siden 80-tallet. For det første har programmering utviklet seg. På 80-tallet foregikk programmering med enkle programmeringstermer som "byggeklosser", med en datamaskin (ofte en sentral stormaskin) og en skjerm. Disse mulighetene er nå kraftig utvidet for eksempel ved at det er mulig å programmere bevegelse (robotikk) og fysiske objekter. Særlig de forskjellige programmerbare datakortene som Micro:Bit, Rasberry Pie og Ardouino har hatt stort gjennomslag. For

[3] http://www.skolesjakken.no/hvorfor-sjakk/

skolen har også utviklingen av enkle grafiske programmeringsomgivelser betydd nye muligheter. Det blir ofte kalt blokkbasert programmering, fordi man kan føye sammen blokker som passer sammen til et program. Da senker man terskelen for å komme i gang med programmering dramatisk, og oversetter det til noe som ligner på å bygge Lego. Den typen programmering kan også gjerne kombineres med de programmerbare datakortene som Micro:Bit og andre.

For det andre har skolen utviklet seg til å bli mer dynamisk. Det er mer plass for prosjektarbeid, og det er flere muligheter for å jobbe tverrfaglig. Det er begge utviklingstrekk som gjør at det er enklere å få plass til programmering i skolen. Videre er det også klart at eleven kommer med en annen erfaringsbakgrunn i bruk av digital teknologi enn elevene hadde på 80-tallet. Samfunnet, organisasjoner og fritiden er blitt mer digitalisert, og elevene blir tidlig brukere av digitale teknologi.

Konteksten for forskning innen programmering i skolen har de siste årene gjerne vært gjort under overskriften "Computational thinking" eller digital tenkning. Wing (2006) starten denne nye bølgen med å etterlyse utdanning i digital tenkning i bred forstand, og med programmering som det viktigste verktøyet for å få det til. Uten noen felles enighet av hva det skal omfatte, har "Computational Thinking" blitt stående som et samlebegrep for den nye bølgen internasjonalt for å innføre programmering i skolen. I en fagbok om temaet fra 2019 innrømmer redaktørene at de ikke har en omforent forståelse av hva begrepet skal inneholde, men kommer likevel med et slags meta-definisjon etter å ha innrømmet uenigheten mellom de forskjellige kapitteltekstene: "they [forfatterene og deltagerne i den tilknyttede workshopen] agree that the clusters of ideas around CT [Computational thinking] is important in a world being increasingly transformed by information technology" (Kong & Abelson, 2019 p5). Voogt et al. (Voogt, Fisser, Good, Mishra, & Yadav, 2015) finner det vanskelig å gi noen definisjon av CT, og mener det heller ikke er formålstjenlig fordi vi trenger fleksibilitet i begrepene nå mens ideene er under utvikling slik at vi ikke innsnevrer mulighetene for videreutvikling av temaet i skolen.

I Wings relansering av begrepet "Computational Thinking" (Wing, 2006) var det fra et tydelig "Computer science"-perspektiv, fra datateknologi, og også lansert i et tidsskrift for nettopp dette faget. I den artikkelen har datateknologi en utvidet betydning, mer i tråd med det som har fått navnet Informatikk i Norge. Tedre og Demming (2016) drar det videre, og lanserer "Computational Thinking" som en "ny og radikalt forskjellig måte å se verden på" (p.125). En måte å karakterisere motsetningene som knytter seg til både innføringen av programmering i skolen og i digital tenkning[4] er at det blant de teknologiorienterte sees på som en endring der metodikk og tenkemåte fra Informatikk blir viktigere, og dekker flere områder av skole, aktiviteter og arbeidsliv. Fra andre perspektiver argumenteres det imidlertid med at utbredelsen av digital teknologi gir behov for andre tenkemåter og innfallsvinkler.

Ved å sortere i de forskjellige synene på "Computational Thinking" (Flórez et al., 2017; Grover & Pea, 2013; Kong & Abelson, 2019; Tedre & Denning, 2016; Voogt et al., 2015; Wing, 2006), og som vi her kan kalle digital tenkning, er det mulig å samle forståelsene av begrepet i noen hovedlinjer:

1. Digital tenkning som forståelse av verden rundt oss. Samtidig som det knytter seg til kompetanser som programmering, algoritmeutvikling, abstrahering og logikk, gjør spredningen av teknologien at digital tenkning blir vår tids hovedmodell for kommunikasjon, samarbeid og analyse av problemstillinger.

2. Digital tenkning som programmeringskompetanse. En relativt spesifikk tenkemåte som drar med seg det å beherske syntaks og semantikk i programmering, men viktigere her er det som gjerne betegnes som pragmatikk; hvordan se muligheter for å utforme digital teknologi til støtte for forskjellige typer oppgaver.

3. Digital tenkning som algoritmisk tekning. Her dreier det seg om en kompetanse i problemløsning i programmering som innebærer dekomponering av et problem i mindre moduler som kan løses trinn for trinn. Utvikling av logiske strukturer, og

[4] Digital tenkning kan være en relativt nøytral norsk oversettelse av «Computational Thinking»

det å kunne dekomponere, abstrahere og aggregere samt arbeide inkrementelt er sentralt. Det kan sees på som en mer spesifikk versjon av punkt 2.

4. Digital tenkning som brede digitale ferdigheter. Det handler om kompetanse i å kunne velge de rette verktøyene til en oppgave eller et problem, og eventuelt tilpasse dette verktøyet hvis det er formålstjenlig. Det dreier seg også om å beherske den digitale verden og å kunne forstå, velge og tilpasse forskjellige typer teknologi til de problemstillingene og utfordringene man møter.

Det er overlapp mellom hovedretningene, men likevel gir de et analytisk verktøy til å diskutere tilnærmingene til programmering i skolen og til digital tenkning.

3. Forskning på undervisningsopplegg

Tradisjonell programmeringsundervisning har vært og er preget av kognitiv læringsteori, altså at læringen som foregår dreier seg om mentale prosesser (Sentance, Waite & Kallia, 2019). Tradisjonelle programmeringsspråk er ofte tekstbaserte og krever at man har kjennskap til språkets "grammatikk" for å produsere noe meningsfylt. Opplæringen følger vanligvis svært strukturerte læringsopplegg, styrt av en lærer/foreleser og bærer preg av at programmering i stor grad startet som et fag i høyere utdanning der dette er en vanlig måte å undervise på. Forelesningene er ofte basert på at elevene/studentene kopierer kode læreren presenterer, mens viktige programmeringskonsepter blir forklart. Denne metoden har også blitt adoptert av lærere som skal undervise elever i grunnskolealder i programmering (Sentance, et al., 2019).

Derimot finnes det en motsats til denne måten å undervise på, og som kjennetegnes av mer ustrukturerte og utforskende læringsaktiviteter som særlig er inspirert av arbeidet ved Massachusetts Institute of Technology (MIT). Sistnevntes arbeid har blant annet ført til utviklingen av det blokkbaserte kodespråket og -miljøet Scratch (Papert, 1980; Resnick, 2017). Scratch er utviklet med formål om å lære barn ned til 8-årsalderen å programmere. Forkjemperne for denne ustrukturerte tilnærmingen, kjent som *konstruksjonisme*, mener

at den kan bidra til utvikling av andre, generelle (ikke fagspesifikke) ferdigheter, slik som kreativitet og problemløsning. Programmeringen er altså en vei til målet, og ikke nødvendigvis det eneste målet i seg selv. Programmeringen skal baseres på de lærende sine egne problemer/ideer, og de skal komme raskt i gang med å lage egen kode uten for mye instruksjon, det viktigste er at de lager noe (derav navnet *konstruksjonisme*). Instruksjon skal i stedet komme etter spørsmål fra elevene. Denne arbeidsmåten krever at man eliminerer den kompliserte grammatikken i de tekstbaserte språkene, og henger nøye sammen med utviklingen av blokkbaserte språk. *Skaperverksteder* (eng. *makerspaces*) benytter ofte denne frie arbeidsmåten i programmeringsopplæring (Bevan, 2017).

3.1. Undervisningsressurser

Å inndele programmeringsundervisning i tekstbasert og strukturert opplæring, og blokkbasert og ustrukturert opplæring er derimot en grov forenkling. TACCLE3 er et Erasmus+-prosjekt[5] som hadde som mål å støtte lærere som underviser elever i blant annet koding på barneskoletrinnet. De fokuserte på å evaluere ulike ressurser (programmeringsspråk, -miljøer, og fysiske verktøy) lærerne kan bruke i programmeringsundervisningen i klasserommet, både blokk- og tekstbaserte. I rapporten blir mange titalls ulike ressurser presentert, som gir et bilde av hvor mange forskjellige tilnærminger det finnes til programmering i skolen. TACCLE3-rapporten har derimot et teknisk perspektiv, og deres merknader om pedagogikk dreier seg for det meste om hvilke klassetrinn de mener ressursene er egnet for, og ikke noe om faktisk pedagogikk. Falkner og Vivian (2015) har gjennomført et lignende prosjekt og fant at selv om det finnes mange gode verktøy så eksisterer det færre ressurser som tar for seg det programmeringsdidaktiske og hvordan lærere skal legge opp undervisningen, eller ta i bruk de mange læringsressursene. Waite (2018) skriver også om denne utfordringen; det er utviklet mange språk og miljøer som lærere kan bruke i undervisningen, men forskning på pedagogiske opplegg er nesten ikke-eksisterende.

[5] http://www.taccle3.eu/en/

3.2. Programmeringspedagogikk

I sin omfattende litteraturgjennomgang av programmeringspedagogikk, kommer Waite (2018) kun med få svar om hva som er effektiv pedagogikk. Imidlertid kommer hun med en rekke forslag til områder man kan studere for å finne mulige svar. Hva som er god pedagogikk er avhengig av ulike programmeringsspesifikke kontekster eller utfordringer som man kanskje ikke finner igjen i andre fag. For eksempel foreslår Waite at vi må utvikle en måte å undervise blokkbasert programmering som gjør overgangen til tekstbasert programmering enklere, og vi må finne ut hvordan man kan støtte elevene til å bevege seg fra en opplæringssituasjon til å klare å bruke det de har lært til å utvikle egne prosjekter. Programmering for fag (f.eks. matematikk) regner Waite som en egen kontekst som krever en egen pedagogisk fremgangsmåte, og som det ikke er gjort mye forskning på.

I stedet for kun å fokusere på spesifikke programmeringskonsepter i undervisningen må vi også lære elevene programmeringsferdigheter (Lye og Koh, 2014). Dog sier forskningen lite om hvilke manglende ferdigheter det er som gjør at mange unge aldri mestrer programmering (Waite, 2018). Lye og Koh (2014) mener i tillegg at elevene må lære om "computational perspectives", altså å ha ulike perspektiver på hva programmering omfatter og innebærer. Opplæringen burde inneholde et bredt spekter av ulike aktiviteter for å dekke alle disse områdene, men det er gjort lite arbeid for å finne ut hva slags aktiviteter og hva slags pensum som egner seg for ulike alderstrinn (Lye & Koh, 2014).

3.3. Mot mer forskningsbasert undervisning i programmering

For å svare på noen av disse utfordringene har Sentance et al. (2019) (inspirert av Lee et al., (2011) sin Use-Modify-Create-modell) utviklet modellen PRIMM[6] (Predict – Run – Investigate – Modify – Make) for å strukturere klasseromsundervisning i programmering uavhengig av språk (tekst- eller blokkbasert) eller andre ressurser. Modellen er ikke en

[6] https://primming.wordpress.com/

komplett didaktisk pakke, men det er en start mot en mer forskningsbasert programmeringsopplæring. PRIMM-modellen er testet ut blant elever i ungdomsskolealder i Storbritannia, med mål om å adressere de mange utfordringene man finner blant begynnende kodere (Sentance et al., 2019). PRIMM skiller seg fra både forelesningsformatet og den ustrukturerte tilnærmingen til for eksempel Resnick (2017). I PRIMM skal elevene ikke kopiere lærerens kode, men strukturert gjennomføre ulike læringsaktiviteter som gradvis beveger seg mot at de skriver egen kode. Et viktig poeng i PRIMM-modellen er at elevene skal lære å uttrykke seg muntlig om programmeringsbegreper og -prinsipper. Lye og Koh (2014) konkluderte i sin review at dette er en viktig egenskap som ofte blir oversett både i programmeringsundervisning og -forskning. Hermans, Swidan og Aivaloglou (2018) fant at elever som er flinkere til å lese kode høyt for andre også presterer bedre som programmerere, og de mener at alle elever burde lære "kodefonologi" - altså å lese kode høyt. Denne tilnærmingen finner vi igjen i Kluge et al. (in press), der elever i norsk ungdoms- og videregående skole produserte "screencasts", eller skjermopptak der de presenterte og forklarte sin egen kode muntlig.

Lye og Koh (2014) mener forskningen på programmering i større grad må ta i bruk "rike" data og kvalitative metoder for å undersøke både elevers skjermaktivitet når de jobber med programmering, og for å få dem til å uttrykke seg muntlig (think-aloud) om hva de tenker.

3.4. Fra blokk til tekst

Selv om blokkbasert kode har blitt en utbredt måte å lære barn og unge programmering så kommer vi ikke unna det faktum at blokkbasert kode nesten utelukkende er et læringsverktøy, og ikke et programmeringsspråk som brukes i industrien. Dersom målet med programmeringen er at elevene skal kunne utvikle kode i "den virkelige verden" så må de kunne skrive tekstbasert kode. Grover, Pea, og Cooper (2015) studerte elevers overgang fra blokk- til tekstbasert programmering og fant ut at det er mulig å gjennomføre på en god måte, og at overgangen bidrar positivt til forståelsen av digitale

verktøy. I kurset deres, "Foundations for Advancing Computational Thinking" (FACT), forsøkte de å skape en balanse mellom strukturerte og ustrukturerte læringsaktiviteter, for å støtte forståelse av programmeringskonsepter uten å gå på bekostning av elevenes muligheter til å drive utforskende læring, i tråd med blant annet Lye og Koh (2014). Likevel finnes det også tekstbaserte språk som er utviklet spesifikt som læringsressurser, også for barn som for eksempel Logo (Papert, 1980). Imidlertid ser det ut til at den blokkbaserte tilnærmingen kommer til å fortsette å være ledende, til tross for at blokker ikke brukes profesjonelt (Kölling, 2015), og mange unge har vansker med overgangen til tekst (Grover et al., 2015).

3.5. Fysisk programmering

Fysisk programmering, eller programmering av fysiske objekter, har blitt stadig mer utbredt i skolen de siste årene, men har sitt utspring i Papert (1980) sine ideer om "objects to think with". Utbredelsen skyldes delvis at teknologien (roboter, kretskort, osv.) har blitt billigere, i kombinasjon med utviklingen av blokkbaserte språk (Waite, 2018). I læreplanen for valgfag i programmering i ungdomsskolen er programmering av fysiske objekter nevnt eksplisitt[7]. Det er utviklet ulike former for fysisk programmering som egner seg helt ned i barnehagealder, og andre former som egner seg for ungdommer/voksne. Innen fysisk programmering finner vi igjen de samme pedagogiske utfordringene som i programmering generelt, men mye fysisk programmering har i tillegg fått kritikk for å være for mannsdominert. Begrunnelsen er at det som skapes tradisjonelt har vært skapt av menn, og at det ikke har vært rom for mer tradisjonell feminin skaping (Kafai et al., 2014).

3.6. Programmering for fag

Ideen bak fysisk programmering er blant annet at det skal bidra til at elever lærer andre (ikke-programmeringsrelaterte) fag. Papert (1980) var en sterk forkjemper for at programmering (både fysisk og på skjerm) bidro til denne formen for læring, men det

[7] https://www.udir.no/kl06/PRG1-01/Hele/Kompetansemaal/programmering

finnes ikke klare tegn på at det faktisk fungerer. Mer forskning på feltet er nødvendig (Waite, 2018). For eksempel fant Mørch, Litherland og Andersen (2019) at elever i alderen 12 til 15 år som jobbet med fysisk programmering ikke nødvendigvis lærte noe om matematikk eller naturfag, men noen elever så at det kunne være en kobling mellom det de lagde og skolefag. Det kan tyde på at denne koblingen må gjøres mer eksplisitt og ikke er noe som oppstår bare ved å la elevene jobbe med (fysiske) objekter som programmeres. Derimot finnes det eksempler som er mer positive til denne tilnærmingen, blant annet ScratchMaths-prosjektet (Benton et al., 2018) som med lovende resultater har latt elever fra 9 til 11 år bruke det blokkbaserte språket Scratch i et eget læringsopplegg spesielt utviklet for å lære matematikk.

4. Sentrale politiske aktørers argumenter for programmering i skolen

I perioden 2011 til 2013 ble det uttrykt kritikk for hvordan digitale ferdigheter ble omhandlet i læreplaner både utenlands og i Norge. Spesielt var det et par taler av sentrale personer fra IKT bransjen (bl.a. Eric Schmidt fra Google) og politikere (bl.a. Michael Gove som var utdanningsminister i Storbritannia) som mente at digitale ferdigheter og datidens læreplaner som fokuserte på bruk av dataprogrammer ikke gjenspeilte fremtidens behov for IKT kompetanse - nemlig produksjon og sikring av datatjenester gjennom programmering. Flere vestlige selskaper slet med rekruttering, og hadde behov for at myndighetene la mer til rette for å utdanne kompetente programmerere.

Estland var tidligst ute med å ta konsekvensene av argumentene, og startet opp programmet ProgeTiger høsten 2012, der de prøvde ut programmering i skolen for alle klassetrinn med full nasjonal utrulling fra høsten 2013 (Sevik et al, 2016). I samme tidsrom kom det også en norsk offentlig utredning fra Digitutvalget (NOU 2013:2) ledet av Torgeir A. Waterhouse som da var direktør for IKT-Norge. Digitutvalget kartla og identifiserte hindre for digital verdiskaping i Norge, og argumenterte for å innføre

programmering i skolen. Samtidig med NOU'en ble "Lær kidsa koding"[8] (LKK) startet, og skulle på frivillig basis gi utstyr og opplæring til lokale kode-klubber rundt omkring i landet[9]. De lokale kode-klubbene skulle på sin side lære opp barn og unge i programmering. Omtrent samtidig med LKK ble nettstedet Code.org lansert, med visjon om at alle elever i alle skoler burde har muligheten til å lære seg programmering. Code.org var støttet av IT-gigantene Google, Microsoft, Amazon og Facebook, og har siden 2013 gjennomført kodetimen (Hour of Code) i en rekke land (Sevik et al., 2016).

I løpet av 2013 til 2015 var det flere Europeiske land som gjorde endringer i sine læreplaner og rullet ut programmering i skolen som bl.a. England, Bulgaria, Danmark, Frankrike, Irland, Israel, Litauen, Malta, Polen, Portugal, Slovakia, Spania, Ungarn og Østerrike. Tiden 2011-2015 kan derfor gjerne kalles den "andre bølgen" med programmering i skolen der det ble klart at internasjonale selskaper, bransjeorganisasjoner, lokale initiativer, ildsjeler, akademikere, og politikere i løpet av et kort tidsrom kom sammen og tok til ordet for innføring av programmering i skolen, og at bølgen denne gangen har fått kraft nok til å materialisere seg som et fag eller en ferdighet som har en rettmessig plass i skolen.

I Norge har myndighetene representert gjennom ulike departementer og Utdanningsdirektoratet (Udir) siden 2013 hatt et økende fokus på programmering i skolen. Utviklingen startet for alvor fra 2011 med oppnevningen av Digitutvalget bestående av en rekke personer fra IKT-bransjen, politiske rådgivere og akademikere. Utvalget ble bedt om å identifisere og kartlegge eventuelle barrierer for digital verdiskaping, og komme med innspill og vurdere tiltak med hensyn til utviklingen innen feltet. Digitutvalget avga sin utredning i januar 2013 (NOU 2013:2) til Fornyings-, administrasjons- og kirkedepartementet, og anbefalte å innføre programmering i skolen. Hovedargumentet var at manglende oppmerksomhet på grunnleggende programmerings- og utviklingsferdigheter gjør at vi utdanner barn og unge til å bruke digitale verktøy, noe som vil gjøre den oppvoksende generasjon til kun konsumenter av digital tjenester.

[8] https://www.kidsakoder.no/
[9] https://www.digi.no/artikler/utrolig-stotte-til-laer-kidsa-koding/288441

Dermed vil vi sakke akterut når det gjelder å se digital tjenesteutvikling og verdiskaping som en foretrukket karrierevei. I tillegg er det viktig at man har nok IKT-forståelse til å kunne ta gode beslutninger i forhold til innkjøp og digitalisering av sine sektorer - feks. i forhold til IKT-sikkerhet. Manglende forståelse av hva som skjer bak skjermen vil svekke grunnlaget for å utvikle sentrale digitale tjenester som Norge trenger i fremtiden.

Programmering var ikke nevnt som tema i forbindelse med utredningen om "Fremtidens skole" i regi av Kunnskapsdepartementet (KD) (NOU 2015:8), og kun delvis nevnt i stortingsmeldingen "Fag – Fordypning – Forståelse" (Meld. St. 28) fra KD. I sistnevnte var argumentet at man ønsket at tilbudet om undervisning i programmering bygges ut fordi det var uklart om læreplanene for grunnopplæringen godt nok dekket den kunnskapen alle borgere må for å kunne delta fullt ut i skole, yrkes- og samfunnsliv, og i forhold til IKT-sikkerhet. I stortingsmeldingen "Digital agenda for Norge" (Meld. St. 27) fra Kommunal- og moderniseringdepartementet, var man imidlertid mer eksplisitt, og la NOU 2013:2 sine vurderinger til grunn for å komme med konkrete forslag fra regjeringen. I stortingsmeldingen gikk regjeringen imidlertid ikke inn for programmering som eget fag, eller som del av andre fag, men som del av et 3-årig pilot- prosjekt med programmering av IKT som valgfag fra skoleåret 2016/2017. Dette ble også iverksatt. I tillegg anerkjente de initiativ og kurstilbud som ble drevet frem av næringsliv og frivillige, som for eksempel LKK.

Temaet ble også diskutert i et notat fra Senter for IKT i utdanningen (Sevik et al., 2016). I notatet følger Senter for IKT i utdanningen argumentasjonen fra Digitutvalget om at man trenger programmering i skolen for fremtidige behov i næringslivet og et stadig mer digitalisert samfunn. Imidlertid dro de debatten også videre i retning av forskning ved å fokusere på fellestrekk mellom ferdigheter for det 21 århundre og algoritmisk tenkning. Ferdigheter for det 21 århundre legger bl.a. vekt på samarbeid, kritisk tenkning, problemløsning, metakognisjon, og grunnleggende digitale ferdigheter. Senter for IKT i utdanningen argumenterte for at disse ferdighetene samsvarte godt med begrepene bak

algoritmisk tenkning[10]. De mente derfor at programmering burde inn i skolen som en del av fagfornyelsen. På kort sikt som en del av eksisterende fag og læreplaner, og på lang sikt som eget fag. Senter for IKT i utdanningen ble senere oppløst, og integrert i Udir der noen gikk over i Udir BetaLab[11] som prøver ut og eksperimenterer med ny digital teknologi i skolen. Programmering i skolen er fortsatt ett av flere temaer de er opptatt av.

Omtrent samtidig med Digitutvalgets rapport i 2013 ble LKK startet og skulle på frivillig basis gi utstyr og opplæring til lokale kodeklubber rundt omkring i landet. Formålet var å støtte kunnskapsmangelen man identifiserte blant barn og unge i Digitutvalgets rapport, ved å bidra til at barn og unge lærer å forstå og beherske sin egen rolle i det digitale samfunnet, og bidra til rekrutteringen til IT-yrkene og realfagene. LKK fikk med seg næringslivsaktører som NHO, fagforeninger som Tekna og NITO, Vitensentrene, og NRK m.fl. som samarbeidspartnere. Per i dag støtter LKK 172 kodeklubber rundt omkring i landet. LKK har også bidratt til gjennomføringen av kodetimen (Hour of Code) der sentrale IT-aktører som Google, Microsoft, Amazon og Facebook har stilt opp med utstyr og personell for å støtte en skoletime med programmering, ferdig tilrettelagt for elever på alle klassetrinn. Kodetimen foregår i desember hvert år, og er svært populær blant norske skoler. I toppåret 2016 meldte over 115.000 elever fra 1242 skoler fra hele landet seg på, men de senere år har tallet gått litt ned.

De 10 ulike vitensentrene i landet støtter også opp om programmering i skolen. De har i flere år støttet populærvitenskapelig opplevelser og læring innen matematikk, naturvitenskap og teknologi der barn og unge får eksperimentere og utforske selv. De samarbeider med Udir og skoleeiere gjennom nasjonale tiltak som for eksempel Skaperskolen[12] og super:bit skole[13] der elevene får prøve seg på programmering, teknologi og håndarbeid på en kreativ og tverrfaglig måte gjennom såkalte skaperverksted. Vitensentrene har ikke noe uttalt argument om programmering i skolen,

---

[10] https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

[11] https://udirbloggen.no/udir-betalab-utproving-eksperimentering-digital-teknologi-laering/

[12] https://skaperskolen.no

[13] https://www.vitensenter.no/superbit/

men mener at elever trenger et visst kjennskap til naturvitenskap og teknologi i et moderne demokrati og at skaperverksted er et bidrag for å lære programmering. Det omfatter både kunnskap som setter de i stand til å ta stilling til aktuelle samfunnsspørsmål og praktisk hverdagskunnskap (f.eks. om bruk av teknologi). Vitensentrene samarbeider også med Naturfagsenteret ved UiO som er et nasjonalt senter for naturfag i opplæringa, og Matematikksenteret ved NTNU som er et nasjonalt senter for matematikk i opplæringa. Både Naturfagsenteret og Matematikksenteret har programmering på agendaen. Sistnevnte arrangerer egne verksted på barn og unge både på småtrinn, mellomtrinn, og ungdomstrinnet.

I akademia har man også over tid vært opptatt av programmering i skolen. I en rapport skrevet av en arbeidsgruppe oppnevnt av Udir bestående av representanter fra UH- og skolesektoren ble det gjort en faggjennomgang av teknologi i grunnopplæringen (inkl. programmering). Arbeidsgruppa foreslo at det skulle opprettes et nytt obligatorisk fag som omfattet teknologi og programmering (Sanne et al., 2016). Det skulle være et praktisk fag der elevene fikk mulighet til å tilegne seg grunnleggende teknologisk kompetanse. Hovedbegrunnelsen for forslaget var at det er et stort misforhold mellom den teknologiske virkeligheten elevene møter, og det de lærer på skolen i dag. Elevene trenger å møte framtidas arbeidsliv og fritid med kompetanse til å håndtere teknologi med kreativitet og forståelse, og de skal oppleve å ha kontroll over sine teknologiske omgivelser. Videre var argumentasjonen delvis skolefaglig - at matematikk og naturfagene i stadig større grad trenger grunnleggende programmeringskompetanse for å løse faglige problemstillinger. Dette behovet ser man spesielt i UH-sektoren der programmering i økende grad brukes i matematikk og naturfagene for å løse problemstillinger rundt numeriske ligninger, og gjøre beregninger for å utvikle modeller og simuleringer. Derfor har Universitetet i Oslo (UiO) i samarbeid med Universitetet i Sørøst-Norge opprettet et nasjonalt senter for fremragende utdanning, Center for Computing in Science Education (CCSE), som tar sikte på å integrere programmering med beregningsorienterte, realistiske problemstillinger for studenter allerede i begynnelsen av universitetsutdanningen. CCSE arbeider tett med Kompetansesenter for

undervisning i realfag og teknologi (KURT) ved det matematisk-naturvitenskapelige fakultet ved UiO som gjennom ProFag tilbyr etter- og videreutdanning for lærere som ønsker å heve programmeringskompetansen i realfagene på både for ungdoms- og videregående skole. ProFag er et samarbeid mellom KURT, Utdanningsetaten i Oslo (UDE) og Akershus Fylkeskommune (AFK), og er finansiert gjennom ordningen for desentralisert kompetanseutvikling (DEKOM).

5. Sentrale aktørers arbeid for å innføre programmering i skolen

KD og Udir foreslår at programmering skal være obligatorisk i fagene matematikk, naturfag, musikk, og kunst og håndverk fra 2020. Programmering vil også fortsette som eget valgfag i ungdomsskolen. For de obligatoriske fagene blir matematikk hovedvertskap for programmering, og allerede fra 2. trinn innføres algoritmisk tenkning gjennom kompetansemålet[14] "lage og følgje reglar og trinnvise instruksjonar i leik og spel". Fra 5. trinn innføres reell programmering gjennom kompetansemålet "lage og programmere algoritmar med bruk av variablar, vilkår og lykkjer", og læreren skal være i dialog med elevane om utviklingen deres i programmering. Etter 9. trinn skal elevene være istand til å bruke programmering til å løse reelle faglige problemstillinger gjennom kompetansemålet "simulere utfall i statistiske fordelingar og berekne sannsynet for at noko skal inntreffe ved å bruke programmering".

For programmering som valgfag skal de nye kompetansemålene[15] sørge for at elevene utvikler algoritmisk tenkning og forståelse av digital teknologi, og at de opplever mestring og kreativitet gjennom utvikling av digitale produkter. Valgfaget baseres på behov fra samfunn og næringsliv (se bl.a. NOU 2013:2), og hente elementer fra matematikk, naturfag, musikk, kunst og håndverk, og samfunnsfag. Kompetansemålene vil fokusere på forstå og forklare koder, feilsøke programmer, bruke og reflektere rundt

[14] https://hoering.udir.no/Hoering/v2/343?notatId=686
[15] https://hoering.udir.no/Hoering/v2/392?notatId=829

grunnleggende prinsipper i programmeringsspråkene, og planlegge og skape et digitalt produkt som er brukervennlig - både individuelt og i samarbeid med andre.

For å få støtte innføringen i den obligatoriske delen med utstyr og kompetanse har Udir tre hovedtiltak. Den ene er satsingen "den teknologiske skolesekken" som bidrar med tilskuddsordninger for utstyr[16], innkjøp av digitale læringsressurser[17], og utvikling av digitale læringsressurser[18] rettet mot skoleeiere, skoleledere og lærere. Den sørger for at lærere og elever har utstyret og de digitale ressursene de trenger for å kunne jobbe med programmering. For programmering ble det satt av 15 mnok årlig fra og med 2018 til og med 2022.

Det andre tiltaket "kompetansepakker" som er en del av "den teknologiske skolesekken". En kompetansepakke[19] er en samling digitale innholdselementer/moduler som presenteres som et strukturert utviklingsløp for et fag. Tidligere har læreboka hatt rollen som et strukturerende element som styrer undervisninga (Gilje et al., 2016), men nå kan det virke som at den får en mindre rolle ettersom en kompetansepakke vil inneholde både struktur og innhold, og være lett tilgjengelig gjennom en digital plattform. På den måten kan man skape en felles forståelse for fagspesifikk undervisning over hele landet ettersom alle lærerne kommer til å ha lik tilgang til samme struktur og innhold. Modulene i kompetansepakkene skal ha en tverrfaglig tilnærming i tråd med fagfornyelsen, og et tydelig fagdidaktisk fokus. Det vil si at den har deler som rettes mot hele skolen, mens andre deler skal rettes mot faglærere i matematikk, naturfag, kunst og håndverk og musikk. Ettersom den er digital vil den over tid kunne støttes og utvikles med struktur og innhold fra kompetansemiljøer med fagrelevant kompetanse. Et eksempel er

---

16 https://www.udir.no/kvalitet-og-kompetanse/nasjonale-satsinger/den-teknologiske-skolesekken/utlysning-av-midler-til-utstyr-for-programmering-i-skolen/

17 https://www.regjeringen.no/no/aktuelt/48-nye-millioner-til-digitale-laremidler-i-skolen/id2643540/

18 https://www.regjeringen.no/no/aktuelt/21-prosjekter-far-stotte-til-digitale-laremidler/id2621353/

19 https://www.udir.no/laring-og-trivsel/lareplanverket/fagfornyelsen/stotte-til-innforing-av-nye-lareplaner/

kompetansepakken for teknologi, programmering og algoritmisk tenkning (TEKPROG) som utvikles av Udir i samarbeid med Universitetet i Oslo, OsloMet og Høgskulen i Volda for å støtte lærere fra 1. til 10. trinn.

Det tredje tiltaket er en ny modell for kompetanseutvikling i skolen kalt DEKOM[20] (noen steder også kalt DEKOMP). DEKOM er en desentralisert ordning som skal bidra til at alle skoleeiere gjennomfører egne lokale kompetanseutviklingstiltak. Fylkesmannen, skoleeiere (kommuner og fylkeskommuner), lokal UH (universitet eller høgskoler) og andre relevante aktører skal samarbeide for å kartlegge behov for kompetanseutvikling, og planlegge utviklingsarbeidet. Staten skal sette overordnede rammer og bidra med finansiering, men skoleeiere må bidra med 30 prosent i egenfinansiering. Skoleeiere kan ikke kjøpe hjelp direkte fra private tilbydere av skoleutvikling, men må opprette kontakt med etablerte UH-miljøer. Fylkesmannen legger til rette for samarbeid mellom skoleeiere og UH-institusjoner der skoleeierne involverer skoleledere og lærere i kartlegging, valg og utforming av tiltak mens UH-institusjonene fungerer som utviklingspartnere for skoleeierne og bidrar med faglig innhold i det lokale utviklingsarbeidet. I Oslo/Viken er det etablert samarbeidsforum med representanter fra skoleeiere, fylkesmannen og UH-sektoren. Skoleeiere og UH-institusjoner kan søke midler og inngå partnerskap etter behov. Askerskolen inngår for eksempel i DEKOM partnerskap med UiO der sistnevnte er koordinert av enheten "Forskning, innovasjon og kompetanseutvikling i skolen" (FIKS)[21]. Kompetanseutviklingen skal være forsknings- og kunnskapsbasert. UH-sektoren skal derfor levere innholdet i kompetanseutviklingstiltakene, og dette er som regel i form av etter- eller videreutdanning. Videreutdanning gir studiepoeng til den enkelte lærer, mens etterutdanning er kompetanseutvikling som ikke gir studiepoeng. Ved UIO gir f.eks. KURT gjennom ProFag etterutdanning for lærere i Oslo (UDE) og Akershus Fylkeskommune (AFK) som ønsker å heve programmeringskompetansen i realfagene[22]. Det vil si at lærerne hever kompetansen i programmering, men får ikke

[20] https://www.udir.no/kvalitet-og-kompetanse/nasjonale-satsinger/ny-modell-for-kompetanseutvikling-i-skole/desentralisert-ordning/desentralisert-ordning/

[21] https://www.uv.uio.no/forskning/satsinger/fiks/

[22] https://www.mn.uio.no/om/samarbeid/tilbud-skoler/kurt/livslang-lering/profag/

studiepoeng. ProFag skal holde flere kursrekker og storsamlinger i løpet av skoleåret 2019/2020. Per dags dato er alle kurs fullbooket av UDE og AFK gjennom DEKOM. OsloMet gir derimot deltids videreutdanning i programmering i skolen for 5. - 10. trinn[23], og for 1. - 7. trinn. Kursene gir 15 studiepoeng hver, og er forbeholdt søkere som er prioritert av Udir gjennom satsingen "Kompetanse for kvalitet".

I tillegg til disse tre tiltakene fra Udir lever enda websiden "IKT i praksis"[24] som ble opprettet av Senter for IKT i utdanningen, og som nå er en del Udir. På denne websiden kan aktører som er opptatt av digitale teknologi i skolen legge ut læringsopplegg for ulike fag. Det har nylig blitt lagt ut flere læringsopplegg i programmering som retter seg mot de nye læreplanene fra kommunale aktører over hele landet.

Når det gjelder næringsliv og frivillige organisasjoner så er det uklart hvilken rolle de har i tilretteleggingen for programmering i skolen. De virker ikke å ha en direkte rolle i DEKOM. LKK har en side med opplegg for programmering som valgfag i perioden 2016 til 2020, men har ikke opplegg for de nye læreplanene som er foreslått. Imidlertid har de vært aktive som del av høringsinnspillene i fag der programmering er aktuelt. I argumentasjonen la de vekt på at programmering ikke kun må handle om å løse problemer i matte- og naturfag, men også legge til rette for skaperglede og utforskertrang gjennom programmering.

Om de store forlagene (Aschehoug, Gyldendal, Cappelen Damm og Fagbokforlaget) blir det nå sagt at de jobber hardt med å lage læremidler for fagfornyelsen. Under SETT konferansen i 2018 på Lillestrøm, og der man fokuserer på å inspirere skoleledere og lærere med ny teknologi og undervisningsopplegg, var det kun 3 av 87 stands som hadde programmering eksplisitt på agendaen. Blant disse 3 var Aschehoug som skrev at de delte ut gratis eksempelhefter og materiell om grunnleggende programmering for både grunnskolen og videregående skole, og at de på nyåret ville invitere til kurs i programmering på alle trinn. Blant annet har de laget eksempelhefter for programmering

[23] https://www.oslomet.no/studier/lui/evu-lui/programmering-skolen-5-10-kompetanse-kvalitet
[24] https://iktipraksis.iktsenteret.no/

i matematikk for 8.-10. trinn[25]. I tillegg tilbyr de gjennom GAN Aschehoug en teknosjekk [26] for lærere som ønsker å søke midler fra Udir til utstyr for programmering i skolen.

6. Oppsummering og diskusjon

For å oppsummere så er det klart at vi nå er inne i den "andre bølgen" med programmering i skolen. Programmering i skolen ble i utgangspunktet sett på som noe som kunne stimulere kognitive ferdigheter lik de vi finner i fagfornyelsen (ferdigheter for det 21. århundre) som for eksempel problemløsning (Papert, 1980). Forskning fra 1980- og 90-tallet tyder imidlertid på at problemløsning i programmering i liten grad lot seg overføre til andre disipliner (Koschmann, 1997). Men siden da har skolen og samfunnet endret seg, og flere av ideene ble relansert gjennom begrepet "computational thinking" (Wing, 2006), som på norsk har blitt oversatt til "algoritmisk tenkning" (Sevik et al., 2016). Imidlertid finnes det ikke en omforent forståelse i forskningslitteraturen om hva begrepet inneholder. Vi forstår begrepet "computational thinking" som bredere enn "algoritmisk tenkning", og foreslår begrepet "digital tenkning" fordi digital teknologi ikke kun handler om kognitive ferdigheter og arbeidsmåter, men også bruk av formålstjenlige verktøy, og at vi tolker og forstår verden i og gjennom digitale verktøy.

For å lære barn og unge programmering finnes det i dag mange undervisningsopplegg som prøves ut. Imidlertid finnes det få effektstudier som kan si om enkelte undervisningsopplegg er bedre enn andre (Waite, 2018). Det som er vanlig er å dele undervisninga i type språk eller type undervisningsopplegg. I hovedsak finnes det to typer språk: blokk- eller tekstbaserte. Det vanligste er å introdusere blokkbasert programmering for yngre elever for så å gå over i tekstbasert (f.eks. fra Scratch til Python), gjerne fra ungdomsskolen (Kölling, 2015). Blokkbasert programmering er lett å begynne med for å lære elevene grunnleggende prinsipper. Forskning antyder at elever som begynner med blokkbasert programmering kan ha lettere for å forstå tekstbasert programmering (se f.eks. Armoni et al., 2015; Grover et al., 2015), men det mangler solid

[25] https://tjenester.lokus.no/open/programmering/nedlasting/matematikk_8_10.pdf
[26] https://gan.aschehoug.no/Aktuelt2/Programmering-i-skolen-Soeke-midler

forskning og kunnskap om dette (Waite, 2018). Undervisningsopplegg deles gjerne i strukturert undervisning (f.eks. tavleundervisning og oppgaveløsning) eller eksplorerende der eleven lærer individuelt eller sammen med andre ved å konstruere objekter (fysiske eller virtuelle) gjennom undersøkelser, og prøving og feiling. Det finnes per i dag lite robust forskning på pedagogiske opplegg for programmering (Waite, 2018), men to opplegg som virker lovende er FACT-metoden som adresserer balansen mellom strukturerte og ustrukturerte læringsaktiviteter (Grover et al., 2015), og PRIMM-modellen som fokuserer på læring gjennom prediksjon, utprøvning, evaluering og endring av kode (Sentance et al., 2019). Begge undervisningsoppleggene fokuserer på variasjon og studentaktive læringsformer. Førstnevnte i form av at læreren legger til rette for lett strukturerte prosjekter der elevene sitter i par, og under veiledning av lærer selv må undersøke hvordan de skal løse problemene gjennom programmering. Sistnevnte i form av samarbeid og muntlighet i kombinasjon med prediksjon, utprøvning, og refleksjon. Begge oppleggene har altså fokus på at elevene selv samarbeider om å undersøke og løse problemstillinger under veiledning av lærer, noe som gir elevene rom til å være engasjerte og kreative. Engasjement i form av deltakelse og samarbeid gir generelt positivt læringsutbytte (Hattie, 2008), og spesielt PRIMM-modellen bør være av interesse for de som underviser i programmering, fordi tilsvarende opplegg i naturfag gir elevene dyp og varig forståelse av naturfaglige prinsipper(Linn & Eylon, 2011).

Programmering integreres i obligatoriske fag som matte-, naturfag, musikk, og kunst-og håndverk fra høsten 2020, og vil fortsatt være et valgfag i ungdomsskolen. Sentrale aktører som har argumentert for å få dette innført er Digitalutvalget (NOU 2013:2), LKK sammen med næringslivsaktører og fagforeninger, Kommunal- og moderniseringdepartementet, Udir i regi av senter for IKT i utdanningen, og deler av akademia. Argumentasjonen følger 3 spor:

- At programmering kan bidra til økte digitale ferdigheter og forståelse som gjør at man kan ta mer informerte beslutninger knyttet til digitalisering av sektorer (NOU 2013:2, Meld. St. 27),

- at programmering som problemløsning overlapper med kritisk tenkning (Sevik et al., 2016), og
- at programmering er en naturlig del av problemløsning i STEM[27]-fag og yrker (Sanne et al., 2016).

Spesielt de to siste argumentene tilsier at programmering kan ha en rettmessig plass i fagfornyelsen. Imidlertid har det vært en diskusjon om programmering skulle være et eget fag eller integrert i eksisterende fag (Sevik et al, 2016; Sanne et al, 2016). I dette spørsmålet landet myndighetene på at programmering skulle integreres i etablerte fag. Vi finner få argumenter for dette, men Udir har nevnt at man ikke ønsket å etablere et eget skolefag[28] samt at de andre nordiske landene har samme modell (Bocconi, Chioccariello & Earp, 2018).

Selve innføringen av programmering i skolen skjer gjennom ulike tiltak. Udir stiller midler til rådighet gjennom "den teknologiske skolesekken" både i form av utstyr til skolene, kjøp og utvikling av digitale læremidler, og digitale kompetansepakker. De har også lansert en modell for kompetanseutvikling i skolen (DEKOM) som er en desentralisert ordning som skal bidra til at alle skoleeiere gjennomfører forskningsbaserte kompetanseutviklingstiltak i samarbeid med lokale universitet og høgskoler - for eksempel i form av etter- og videreutdanning av lærere i programmering.

Måten innføringen gjennomføres tyder på at Udir og UH-sektoren samarbeider tett om dette, og det fremstår uklart hvilken rolle det private næringsliv og frivillige har i denne satsingen. Det kan virke som at de to sistnevnte må operere som tidligere gjennom støtteordninger i regi av Udir der næringslivsaktører kan gå sammen med skoleeier (kommunen) og søke midler til lokal utvikling av digitale læremidler eller undervisningsopplegg, eller ordninger i regi av forskningsrådet der næringslivsaktører går sammen med forskere for å prøve ut ny teknologi eller undervisningsopplegg i skolen. Det er også mulig at den nye digitale portalen Udir virker å opprette med tanke på

[27] Science, technology, engineering and mathematics
[28] https://udirbloggen.no/kjerneelementer-i-matematikk-men-hvorfor-programmering/

"kompetansepakker" kan være en inngang for private aktører som utvikler digitale læringsressurser og undervisningsplaner til fag der programmering innføres.

Et aspekt å være oppmerksom på er at tilrettelegging for programmering i skolen har gått relativt raskt sett i lys av kunnskap vi har om faget relativt til andre fag i grunnskolen, og at det i liten grad foreligger evalueringsresultater fra andre land som innførte dette tidligere. Imidlertid har det begynt å komme resultater fra Storbritannia der programmering ("Computing") ble innført som eget fag i 2012. Foreløpige rapporter tyder på at antall timer med programmering har falt drastisk siden 2012, og mange STEM lærere slutter fordi arbeidsbyrden blir for stor eller at de får jobb i IT-sektoren (The Royal Society[29], 2017). Dette er naturligvis bekymringsfullt både med tanke på om man klarer å gjennomføre programmering i norsk skole, men også med tanke på at dersom matte- og naturfagslærere slutter på grunn av forventninger og arbeidspress så kan det underminere deler av fagfornyelsen. I Norge innføres ikke programmering som eget fag (bortsett fra i valgfag), men samtidig mangler mange norske matte- og naturfagslærere programmeringskompetanse som de nå må tilegne seg, og bruke i klasserommet. Siden det for disse lærerne fremstår som ukjent fagstoff vil det potensielt gi en ekstra byrde i forhold til planlegging og gjennomføring, noe som det vil ta tid før vi vet konsekvensene av. Selv om skoleeiere og UH-sektoren bidrar med ressurser til å undervise lærere i programmering vet man lite om kvaliteten er god nok, og favner alle lærerne omkring i landet som trenger kompetansen. Å lære seg programmering i form av å mestre språk og praksis er noe man må jobbe med kontinuerlig over lang tid, gjerne med reelle og engasjerende problemstillinger, og sammen med andre. Det er ikke enkelt å få til som etterutdanning i tillegg til lærernes ordinære arbeid, fritid og familieliv. Rundt denne problemstillingen kan det tenkes at private aktører kan tilby kurs til lærerne, enten fysisk eller nettbaserte, som fokuserer på grunnleggende prinsipper i programmering, bruk av ulike type programmeringsverktøy, og ikke minst skreddersydde undervisningsopplegg for de ulike trinnene i Norge.

[29] https://royalsociety.org/topics-policy/projects/computing-education/

7. Referanser

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "rea" programming. *ACM Transactions on Computing Education (TOCE), 14*(4), 25.

Benton, L., Saunders, P., Kalas, I., Hoyles, C., & Noss, R. (2018). Designing for learning mathematics through programming: A case study of pupils engaging with place value. *International journal of child-computer interaction, 16*, 68-76.

Bevan, B.: The promise and the promises of making in science education. *Studies in Science Education 53*(1), 75-103 (2017).

Bocconi, S., Chioccariello, A. and Earp, J. (2018). *The Nordic approach to introducing Computational Thinking and programming in compulsory education.* Report prepared for the Nordic@BETT2018 Steering Group. Available at: http://www.itd.cnr.it/doc/CompuThinkNordic.pdf

Bork, A. (1981). *Learning with computers*. Bedforms. Mass.: Digital press.

Brinkley, M., Erstad, O., Herman, J., Raizen, S., Ripley, M., Miller-Ricci, M., & Rumble, M. (2012). Defining Twenty-First Century Skills. In P. Griffin, B. McGaw, & E. Care (Eds.), *Assessment and Teaching of 21st Century Skills*.

Falkner, K., & Vivian, R. (2015). A review of computer science resources for learning and teaching with K12 computing curricula: An Australian case study. *Computer Science Education, 25*(4), 390–429.

Flórez, F. B., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research*, *87*(4), 834–860. https://doi.org/10.3102/0034654317710096

Gilje, Ø., Ingulfsen, L., Dolonen, J., Furberg, A., ..., Granum, K. (2016). *Med ARK&APP. Bruk av læremidler og ressurser for læring på tvers av arbeidsformer*. Universitet i Oslo, Norway. https://www.udir.no/tall-og-forskning/finn-forskning/rapporter/ark-og-app-i-norske-klasserom/

Grover, S., & Pea, R. (2013). Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43. https://doi.org/10.3102/0013189X12463051

Grover, S., Pea, R. & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education, 25*(2), pp.199–237.

Hattie, J. (2008). *Visible Learning*. Abingdon, Oxon: Routledge.

Kafai, Y., Fields, D. Searle, K.. (2014). Electronic Textiles as Disruptive Designs: Supporting and Challenging Maker Activities in Schools. *Harvard Educational Review. 84*. 532-556. 10.17763/haer.84.4.46m7372370214783.

Kalelioğlu, F. (2019). Characteristics of Studies Conducted on Computational Thinking: A Content Analysis. In M. S. Khine (Ed.), *Computational Thinking in the STEM Disciplines: Foundations and Research Highlights*. Springer International Publishing.

Kluge, A., Litherland, K. T., Borgen, P. H. & Langslet, G. O. (In press). Speaking in Codes: Combining programming with audio explanations.

Kong, S. C., & Abelson, H. (2019). *Computational thinking*. Singapore: Springer imprint.

Koschmann, T. (1997). Logo-as-Latin Redux. *The Journal of the Learning Sciences*, *6*(4), 409–415. Retrieved from http://www.jstor.org/stable/1466780

Kölling, M. (2015). Lessons from the Design of Three Educational Programming Environments: Blue, BlueJ and Greenfoot. *International Journal of People-Oriented Programming (IJPOP), 4*(1), 5–32.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., Malyn-Smith, J., et al. (2011). Computational thinking for youth in practice. *ACM Inroads, 2*(1), 32–37.

Linn, M., & Eylon, B. S. (2011). *Science learning and instruction: Taking advantage of technology to promote knowledge integration*. New York, NY: Routledge.

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.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to Program and Learning to Think: What's the Connection? *Commun. ACM*, *29*(7), 605–610. https://doi.org/10.1145/6138.6142

Meld. St. 27 (2015–2016). Regjeringen. (n.d.). *Digital agenda for Norge — IKT for en enklere*

hverdag og økt produktivitet

Meld. St. 28 (2015-2016). Regjeringen. (n.d.). *Fag – Fordypning – Forståelse En fornyelse av Kunnskapsløftet.*

Mørch A.I., Litherland K.T., Andersen R. (2019) End-User Development Goes to School: Collaborative Learning with Makerspaces in Subject Areas. In: Malizia A., Valtolina S., Morch A., Serrano A., Stratton A. (eds) *End-User Development*. IS-EUD 2019. Lecture Notes in Computer Science, vol 11553. Springer.

Nickerson, R. (1983). Computer programming as a vehicle for teaching thinking skills. *Thinking: The Journal of Philosophy for Children*, *4*(2/3), 42–48.

NOU 2013:2. (2013). *Hindre for digital verdiskaping*. Retrieved from https://www.regjeringen.no/no/dokumenter/nou-2013-2/id711002/

NOU 2015:8. (2015). *Fremtidens skole : Fornyelse av fag og kompetanser*. Retrieved from https://www.regjeringen.no/no/dokumenter/nou-2015-8/id2417001/

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. Basic books.

Papert, S. (1993). *The Children's Machine: Rethinking School in the Age of the Computer.* New York: Basic books.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137–168. https://doi.org/https://doi.org/10.1016/0732-118X(84)90018-7

Resnick, M. (2017). *Lifelong Kindergarten: Cultivating creativity through projects, passion, peers, and play.* Cambridge, MA: MIT Press.

Salomon, G., & Perkins, D. N. (1987). Transfer of Cognitive Skills from Programming: When and How? *Journal of Educational Computing Research*, *3*(2), 149–169. https://doi.org/10.2190/6F4Q-7861-QWA5-8PL1

Sanne, A., Berge, O., Bungum, B., Jørgensen, E. C., Kluge, A., Kristensen, T. E., … Voll, L. O. (2016). *Teknologi og programmering for alle:En faggjennomgang med forslag til endringer i grunnopplæringen*. Retrieved from https://www.udir.no/globalassets/filer/tall-og-forskning/forskningsrapporter/teknologi-og-programmering-for-alle.pdf

Sevik, K. m. fl. (2016). *Programmering i skolen*. Notat fra Senter for IKT i utdanningen, november 2016. Retrieved from https://www.udir.no/globalassets/filer/programmering_i_skolen.pdf

Sentance, S., Waite, J. & Kallia, M. (2019) Teaching computer programming with PRIMM: a sociocultural perspective, *Computer Science Education, 29*:2-3, 136-176, DOI: 10.1080/08993408.2019.1608781

Soloway, E., Lochhead, J., & Clement, J. (1982). Does Computer Programming Enhance Problem Solving Ability? Some Positive Evidence On Algebra Word Problems. Research supported in part by NSF Grant SED 78-22043 in the Joint National Institute of Education — National Science Foundation Program of. In R. J. Seidel, R. E. Anderson, & B. Hunter (Eds.), *Computer Literacy* (pp. 171–201). https://doi.org/https://doi.org/10.1016/B978-0-12-634960-3.50023-3

Tedre, M., & Denning, P. J. (2016). The Long Quest for Computational Thinking. *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*, 120–129. https://doi.org/10.1145/2999541.2999542

The Royal Society (2017). *After the Reboot – Computing Education in UK Schools*. Royal Society report available at: https://royalsociety.org/topics-policy/projects/computing-education/

Thorndike, E. L. (1923). The Influence of First-Year Latin Upon Ability to Read English. *School & Society*, *17*, 165–168.

Voogt, J., Fisser, P., Good, J., Mishra, P., & Yadav, A. (2015). Computational thinking in compulsory education: Towards an agenda for research and practice. *Education and Information Technologies*, *20*(4), 715–728. https://doi.org/10.1007/s10639-015-9412-6

Waite (2018). *Pedagogy in teaching Computer Science in Schools: A Literature Review*. Lastet ned 10.10.2019 fra https://royalsociety.org/computing-education

Wing, J. M. (2006). Computational Thinking. *Commun. ACM*, *49*(3), 33–35. https://doi.org/10.1145/1118178.1118215