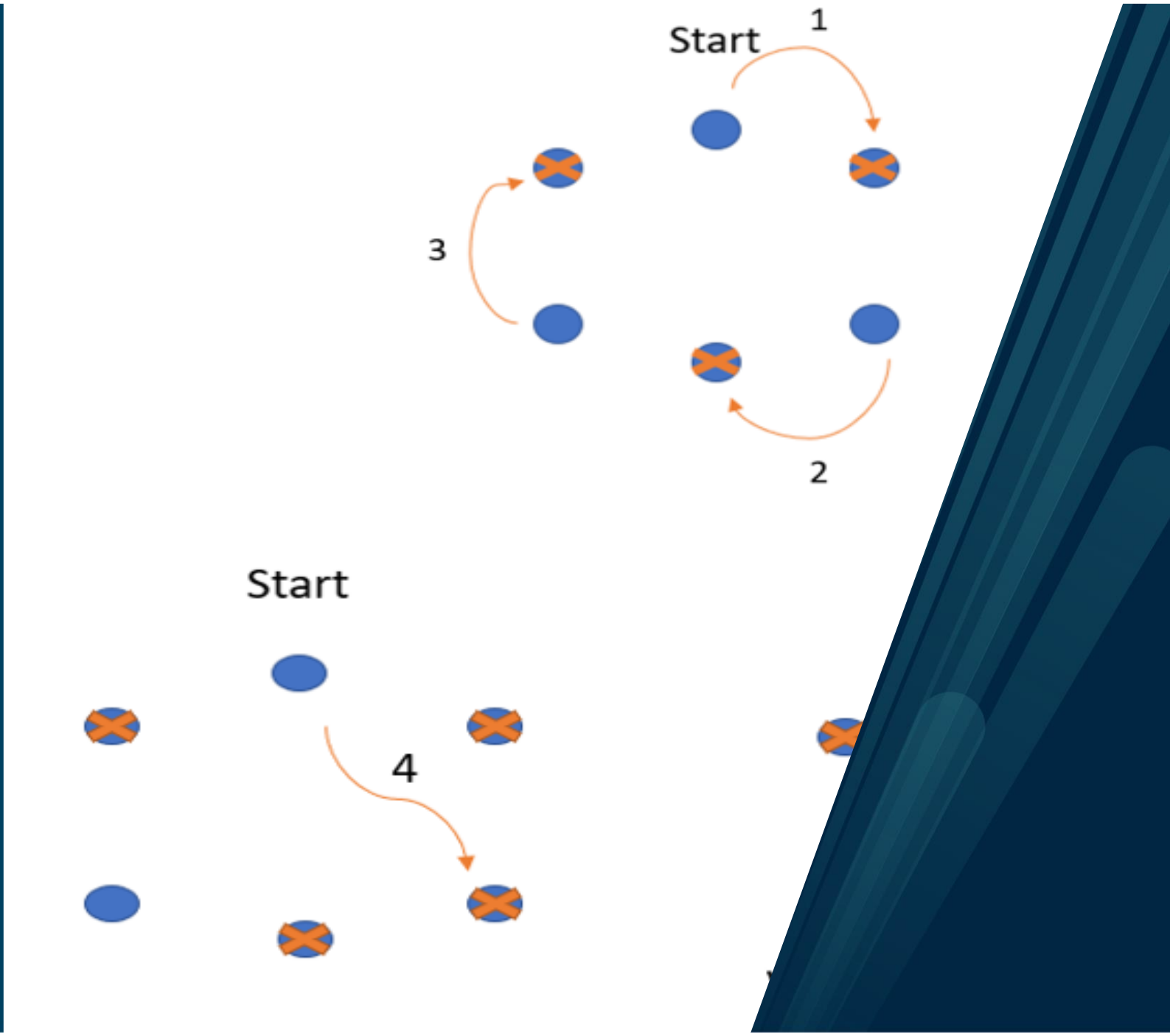
UiT Norges arktiske universitet

Institutt for matematikk og statistikk

Hvordan en programmerer løser matematiske problemer

Birk Eirikssønn Heiberg

Masteroppgave i Lektor i realfag (MAT-3907), juni 2022



Innholdsfortegnelse

Figurliste 3

1 Innledning 1

2 Teori 4

2.1 Hva som menes med et problem 4

2.2 Problemløsning i matematikk 5

2.3 Problemløsning som en kunst 6

2.4 Gestalt-modellen 7

2.4.1 Kreativitet i problemløsning 8

2.5 Lesters syn på problemløsning 9

2.6 Schoenfelds problemløsningsmodell 10

2.7 Programmering 11

2.8 Algoritmisk tenking 13

2.8.1 Algoritmisk resonnering 15

2.9 Programmering i matematikkundervisning 16

2.9.1 Programmering i svensk skole 16

2.9.2 Effekter av å undervise programmering parallelt med matematikk 18

3 Metode 19

3.1 Case-studie 19

3.2 Josephus Problem 20

3.3 Begrunnelse for valg av problemet 22

3.4 Aktive samtalebaserte intervju 22

3.5 Josephus problem i intervjuet 23

3.6 Forskerens rolle 23

3.7 Spørsmålene i intervjuet 24

3.7.1 Pilotering av intervjuet 25

3.8 Informanten 26

3.9 Forskningsetikk 27
3.10 Analysemetode 28
3.11 Kvalitet 28
Data og 31
4 analyse 31
4.1 Josephus problem 31
4.1.1 Problemet i et runde-tankesett 33
4.1.2 Problemet i et rest-tankesett 34
4.1.3 Problemet brutt ned i delproblemer 36
4.1.4 Analyse av Josephus problem med Schoenfelds modell 39
4.2 Spørsmålene i intervjuet 41
4.2.1 Pers bruk av begrepet programmering 41
4.2.2 Likheter mellom problemløsing i programmering og matematikk 42
4.2.3 Ulikheter i problemløsing i programmering og matematikk 44
4.3 Konklusjon 46
4.4 Fremtidig arbeid 47
Referanseliste 49
Vedlegg 52

Figurliste

Figure 1:Problemløsningsprosessen til uerfarne problemløsere, ifølge Schoenfeld (1992) 11
Figure 2: "To create Scratch programs, users snap together graphical blocks". From andresmh, CC BY-SA 2.0. https://www.flickr.com/photos/amonroy/2250306046 13
Figure 3: Josephus problem med 6 krigere 21
Figure 4: Problemløsningsprosessen til Per 39
Figure 5: Tidsbruken til Per i Schoenfelds (1992) modell 40

Forord

Tusen takk til min hovedveileder Anne Birgitte Fyhn for mange gode innspill og diskusjoner under dette prosjektet. Takk til min biveileder Ragnar Soleng for innspill på teksten. Oppgaven hadde vært fattig for innhold uten min informant, så tusen takk til deg!

Videre må jeg gi en takk til alle mine medstudenter, spesielt realfagsgjengen som det har vært herlig å studere med i 5 år. Tusen takk til mine foreldre, Ina og Eirik, for å ha stilt opp med korrekturlesing og for mange fine stunder gjennom hele masterskrivingen. Takk til mine søsken, venner og min kjæreste for alle de gode timene. Takk til blant annet Coheed & Cambria, Pink Floyd og Between & Buried and me for god musikk under masterskrivingen.

Sammendrag

Programmering og problemløsing ble introdusert i matematikkfaget i fagfornyelsen (KD, 2020). Begge disse skulle være overordnede temaer som skulle brukes gjennom matematikkundervisningen. Hvordan arbeid med programmering påvirker problemløsing i matematikk er derfor noe som elevene ville bli direkte påvirket av. Denne relasjonen var det som motiverte utformingen av problemstillingen «Hvordan løser en programmerer matematiske problemer?». Det er flere forskjellige teorier på hvordan problemløsing foregår i matematikk, og noen av disse blir redegjort for. Videre drøftes det hva programmering er og hvordan programmering relaterer til algoritmisk tenking. Metoden for å undersøke problemstillingen var å la en programmerer løse et matematisk problem kjent som Josephus problem og et intervju i etterkant av dette. Hovedfunnet i denne oppgaven var at en programmerer har mange likhetstrekk med hvordan matematikere løser matematiske problemer. Programmereren selv mener at problemløsing i matematikk og programmering har samme overordnede prosess. Det argumenteres for i denne oppgaven at det å kunne programmere kan fostre gode problemløsingsvaner i matematikk. Dette kan bety at det er gjort et godt valg med å implementere både programmering og problemløsing samtidig i fagfornyelsen.

1 Innledning

I Mønsterplanen av 1987, M87, (Kirke- og undervisningsdepartement, 1987) ble for første gang datamaskinbruk eksplisitt introdusert i matematikkfaget. Datamaskinbruk har etter dette vært inkludert i matematikkfaget. I fagfornyelsen er det eksplisitt påpekt at programmering skal integreres i matematikkfaget (Kunnskapsdepartementet [KD], 2020). M87 og fagfornyelsen har noen likhetstrekk, da problemløsning og det å bruke datamaskin er overordnede temaer i begge disse. Det er derfor interessant å se på forholdet mellom programmering og problemløsing i matematikken. Er det et positivt samspill mellom dem? Kommer kompetanse innen programmering på bekostning av kompetanse innen matematikk? Dette er sentrale spørsmål å stille seg, men det er betraktelig vanskeligere å svare på disse spørsmålene enn å stille dem. De fleste lærere har ingen formell utdanning innenfor programmering. De som derimot jobber med programmering som yrke sitter med unik kunnskap om programmering og deres innblikk kan gi verdifulle innspill til hvordan programmering kan brukes i skolen.

M87 la stor vekt på problemløsning i matematikkfaget, og det skulle være et hovedemne i faget. Dette betydde at problemløsning skulle være en del av all matematikkopplæring. Begrunnelsen for det var at elevene gjennom det skulle utvikle tenke- og arbeidsvaner som ville gi en positiv utvikling for eleven. Videre ble det påpekt at erfaringene kunne bli til verdifull kunnskap for elevene. At elevene får formulere sine egne problemer vil gi elevene evne til å bruke matematikk som et verktøy for å stimulere deres kreativitet, ifølge M87.

Problemløsning ble beskrevet som en fire-stegs prosess i M87.

1. Å formulere problemet
2. Å analysere problemet og komme fram til en løsningsmetode
3. Å foreta de nødvendige beregninger
4. Å vurdere framgangsmåte og resultater

Elevene skulle arbeide med å formulere egne problemer både muntlig og skriftlig. Det skulle også bli gitt spesielt mye oppmerksomhet på at elevene skulle komme fram til løsningsmetoder når problemet var gitt i form av tekst. Disse løsningsmetodene skulle kunne presenteres på forskjellige måter og kunne presentere oversiktlig. Videre skulle elevene kunne vurdere resultatet og framgangsmåtene brukt og begrunne hvorfor de er rimelige. Det ble også påpekt at overslagregning skulle være en naturlig del av arbeidet med problemløsning.

Ifølge M87 skulle datamaskin være et verktøy for å illustrere matematikk og for å utforske sammenhenger i matematikken. Datamaskinen skulle brukes innenfor alle hovedemnene i faget. For å lære elevene om bruk av datamaskin i matematikkfaget foreslår M87 at datamaskinen burde brukes områder der det er spesielt egnet å bruke. Datalære er basert på algoritmer, og det påpekes at datalære har en stor plass i hovedemnet problemløsning.

Til forskjell fra senere læreplaner, var mønsterplanen M87 bare et veiledende dokument til lærerne. Dette betydde i praksis i at lærerne kunne se bort ifra store deler av mønsterplanen hvis de ønsket dette. Mønsterplanen oppfordret også i stor grad til lokalt læreplanarbeid, der lærerne som kollektiv skulle utforme lokale planer. I hvor stor grad det ble arbeidet med problemløsning og med datamaskin i matematikk var derfor opp til den enkelte lærer og skole å bestemme.

I fagfornyelsen fra 2020 er det påpekt at programmering og problemløsing skal være en del av elevers opplæring. Elever vil derfor i matematikk bli eksponert for arbeid med både programmering og problemløsing. Samspillet mellom programmering og problemløsing vil påvirke elever under den nye læreplanen.

Jeg hadde ikke brukt programmering i noen særlig grad før jeg tok matematikk- og fysikk-emner på universitet. Jeg fant fort ut av programmering var noe jeg likte godt og som hadde mange bruksområder innen matematikk og fysikk. Samtidig følte jeg at måten å tilnærme seg problemene innenfor programmering hadde noe eget med seg, som også påvirket måten jeg løste problemer innenfor matematikk og fysikk.

Når man løser problemer med bruk av programmering, er det betydelig mer arbeid med å tenke over hvordan en funksjon faktisk skal implementeres og hva den skal gjøre til enhver tid. Det brukes mye tid på dette og det kommer naturlig å tenke hvordan dette kan forenkles. Da må man tenke over hva som faktisk kreves at denne funksjonen, noe som kan gi en dypere forståelse av funksjonen og også problemet det arbeides med.

Programmering i matematikkfaget etter den nye læreplanen vil påvirke elevene, og mine personlige erfaringer tilsier at dette kan være en positiv ting. På bakgrunn av dette ble følgende problemstilling for masteroppgaven formulert «Hvordan løser en programmerer matematiske problemer?»

Oppgaven er bygd opp slik at i neste kommer teori om problemløsing i matematikk og programmering både generelt og i matematikk. Deretter i neste kapittel vil det bli redegjort for metodevalg for innsamling av data og hvordan dataen vil bli analysert, i tillegg vil det også bli redegjort for informantvalg. Til slutt følger analyse av dataen og en diskusjon av dette. Til slutt kommer konklusjonen av arbeidet.

2 Teori

Dette kapitlet har tre hoveddeler. Begrepene programmering og problemløsing blir drøftet, da disse er forutsetninger for å forstå hva som menes med problemstillingen. Videre blir litteratur om bruk av programmering i matematikk redegjort for. Hvordan programmering har blitt brukt i skolen, og hvordan det nå blir brukt, blir også presentert. Til slutt blir relasjoner mellom problemløsning og programmering gjennomgått.

2.1 Hva som menes med et problem

Problemløsning er et sammensatt begrep. For å kunne forstå problemløsing er det nødvendig å forstå hva som menes med et problem.

Root-Bernstein (2003) klassifiserer problemer i henhold til deres *type, klasse og orden. Typen* til problemet sier noe om metoden som må brukes for å løse problemet (Root-Bernstein, 2003). Metodene for å løse problemet «Hva er fart?» og problemet «Hvilken modell er best for å predikere været i Tromsø?» vil ikke være de samme.

Klassen til problemet sier hvorvidt problemet er løsbart generelt, bare i spesialtilfeller eller uløselig. Å vite klassen til problemet kan si noe om hvor viktig problemet er, da problemer som er løsbare for store områder vil relatere til andre problemer i større grad (Root-Bernstein, 2003).

Ordenen til problemet sier hvordan problemet relaterer til andre problemer. Høy-ordens problemer relaterer til mange andre problemer, mens lavere-ordens problemer relaterer til færre problemer (Root-Bernstein, 2003).

Motivasjonen for å klassifisere og evaluere problemer er blant annet å kunne konstruere problemtrær. Disse problemtrærne kan brukes for å undersøke hvilke problemer som ikke kan løses enda og hvilke andre problemer som må løses først for å ha mulighet til å løse det gitte problemet (Root-Bernstein, 2003). Ved å gjøre dette er det mulig å bestemme hvor viktig et gitt problem er å løse. Et problem som tidligere ble vurdert som uviktig kan bli kjempeviktig hvis det er relatert til et annet problem som vurderes som viktig (Root-Bernstein, 2003).

I denne oppgaven er det matematiske problemer som er av størst interesse å diskutere. Ifølge Schoenfeld (1992) har det tradisjonelt vært to definisjoner som har blitt brukt for å beskrive et problem i matematikk. Disse definisjonene er som følger

"*Noe som kreves å gjøre"* (Min oversettelse fra "*In mathematics, anything required to be done, or requiring the doing of something*" (Schoenfeld, 1992 s. 10))

"Et spørsmål, som er krevende eller forvirrende" (Min oversettelse fra *"A question...that is perplexing or difficult"* (Schoenfeld, 1992, s.10))

Hvilken definisjon et problem i matematikk forstås ut ifra har betydning både for undervisning i problemløsing i matematikk og hvordan problemløsing i matematikk foregår.

2.2 Problemløsning i matematikk

Stort sett har problemløsing i matematikk blitt tolket på tre forskjellige måter gjennom tidene (Schoenfeld, 1992).

- Problemløsing som en kontekst
- Problemløsing som en ferdighet
- Problemløsing som en kunst

Den første definisjonen har lang historie. En slik definisjon kan finnes i William Milnes verk *A mental arithmetic* fra 1897. I dette verket er hensikten med problemene som gis at elevene skal lære seg en bestemt teknikk (Schoenfeld, 1992). Milnes problemer faller under den første definisjonen av et problem i matematikk. Et eksempel på dette er følgende problem med løsning

- Problem: How much will it cost to plow 32 acres, when the cost per acre is 3.75 $?
- Løsning ifølge Milnes bok: 3.75 $ er 3/8 av 10 $. Hvis prisen på hver acre var 10 $ ville all pløyingen kostet 320 $, men siden den faktiske prisen er 3.75 $ er den faktiske totale prisen 3/8 av 320 $ noe som blir 120 $.

Den generelle strukturen for presentasjonen av Milnes problemer er: 1) elevene blir gitt et problem for å introdusere en bestemt teknikk, 2) teknikken blir demonstrert av læreren, 3) elevene blir gitt flere lignende problemer for å øve på teknikken.

Det er flere vanskeligheter med Milnes tilnærming til problemer og problemløsing i matematikk (Schoenfeld, 1992). Problemene som skal gjennomføres er kunstig framstilt. Det vil si at tallene i oppgavene (billettpriser, fruktpriser, åkerpløyingspris, etc.) er ikke virkelige og kun valgt for å gjøre matematikken i oppgaven mer lettvint å gjennomføre. Som

konsekvens av dette flyttes matematikken fra den virkelige verden til en *kunstig* verden, en verden elevene vanskelig kan kjenne seg igjen i.

Schoenfeld hevder videre at en annen utfordring følger av at de konstruerte problemene kun er laget for å lære bort én teknikk. I oppgaven om prisen for å pløye en åker er det flere mulige måter å gå fram for å løse denne oppgaven. Man kunne for eksempel ha regnet 4 $ ganger 32 og så trukket fra 0.25 ganger 32. Dette tar ikke noe lengre tid og vil kanskje være mer nærliggende for enkelte elever. Når man lærer bort kun én teknikk for et gitt problem kan elevene få en oppfatning av at en gitt type oppgave kun har en løsningsmetode. Videre fører dette til at elevene aldri får prøve seg fram til å finne løsningsmetoder på egen hånd. Dette kan gi elevene inntrykk av at de ikke skal lære seg metoder på egen hånd. Når man bruker problemer for å lære bort spesifikke teknikker, kan man skape et feilaktig inntrykk av hva matematikk er. Med denne tilnærmingen reduseres matematikken til en liste av diverse teknikker og fakta.

Problemløsing som en ferdighet kom som reaksjon på forskningen til Thorndike (1901). Thorndike forsket på problemløsing innen matematikk og viste at det å være en god problemløser innen matematikk ikke nødvendigvis betydde at en var en god problemløser i andre områder. I denne tolkningen av problemløsing er det å være en god problemløser verdifullt i seg selv.

2.3 Problemløsning som en kunst

Problemløsing som en kunst stammer fra verket “How to solve it?” av Polya (1957). Mye av forskningen på problemløsning bruker Polyas modell som utgangspunkt.

Polya (1957) bryter problemløsing ned i fire steg. Det var akkurat denne typen framgangsmåte som skulle bli brukt i M87.

1. Å forstå problemet
2. Å lage en plan
3. Å utføre planen
4. Å se tilbake på problemet

Det første steget handler om å sette seg i detalj inn i hva problemet innebærer. Hva er den ukjente? Hva er forutsetningene? Hva er begrensningene? Man må se på problemet fra flere sider og ha full oppmerksomhet på det.

Å lage en plan er den lengste og mest utfordrende delen av å løse et problem. Når man lager planen må man være bevisst på problemets avgrensninger, hvilke betraktninger man må gjøre og hvilke verktøy man kan anvende. For å kunne lage en plan som kan løse problemet kreves det at problemløseren har kunnskap på det matematiske området som problemet befinner seg i. Å lage en plan som vil lykkes krever at problemløseren får en god ide. Det å få en god ide er ikke alltid enkelt. Polya foreslår at problemløseren kan stille seg selv (eller en elev) følgende spørsmål for å komme på rett spor: Kjenner vi til et relatert problem? Kjenner vi til et annet problem med den samme ukjente? Polya påpeker også at hvis vi ikke klarer å løse problemet, så finnes det kanskje et enklere, men relatert problem som vi kan løse. Det kan være fordelaktig å forsøke å løse dette problemet først.

Når man har laget planen for problemløsning går man videre med å utføre planen. Det som er viktig i utføring av planen er at problemløseren er overbevist om at hvert steg som gjennomføres er korrekt. Denne overbevisningen kan stamme fra intuisjon eller fra formelle bevis. Det som kan gå galt i denne fasen av problemløsing er at problemløseren glemmer planen sin. Det skjer oftest hvis planen stammer fra noen andre enn problemløseren selv.

Å se tilbake på problemet er et viktig steg som ifølge Polya ofte ikke blir gjennomført. Det er to formål med dette steget. Det ene er å bekrefte at løsningen er korrekt. Hvis problemløseren kommer fram til at løsningen er korrekt, kan vedkommende undersøke om man kunne løst problemet annerledes eller om metoden kan brukes på andre problemer. Det vil gjøre problemløseren bedre i stand til å løse problemer i framtiden.

2.4 Gestalt-modellen

Et annet syn på problemløsning og kreativitet er kjent som Gestalt-modellen. Denne modellen har også fire steg (Haavold & Sriraman, 2021).

1. Forberedelse (Preparation)
2. Inkubering (Incubation)
3. Illuminasjon (Illumination)
4. Verifikasjon (Verification)

I det første steget i Gestalt-modellen arbeider problemløseren aktivt med å forstå problemet. Når problemløseren når et punkt der de ikke klarer å få fremgang legges problemet til sides. Da er problemløsingen i inkuberingsfasen. I denne fasen jobbes det ikke aktivt med problemet. Problemløseren gjør da andre ting som muligens er urelatert til problemet. De som

følger Gestalt-modellen påpeker at underbevisstheten jobber med problemet i dette stadiet (Haavold & Sriraman, 2021).

Etter en tid i inkuberingsfasen dukker plutselig en potensiell løsning opp, et slags «Eureka!-øyeblikk». Prosessen før problemløseren når dette øyeblikket kan ta dager, uker eller år. Deretter følger verifikasjonssteget. I dette steget sjekkes det om den potensielle løsningen er korrekt. Er den ikke det, må problemløseren gå tilbake til inkuberingsfasen. Er løsningen derimot korrekt, må løsningen presiseres og formaliseres. Deretter undersøker problemløseren om løsningen kan brukes i andre sammenhenger (Haavold & Sriraman, 2021).

I kontrast med Polyas modell (Polya, 1957) som har blitt brukt i læreplaner (Kirke- og undervisningsdepartementet, 1987) har ikke Gestalt-modellen fått mye plass i læreplaner. Dette skyldes muligens at inkubering er noe som tar lang tid og som det ikke er rom til i en travel skolehverdag.

2.4.1 Kreativitet i problemløsning

Det er betydelig forskjell på hvordan disse modellene forklarer hvordan man kan få innsikt i et problem (Haavold & Sriraman, 2021). Polyas modell legger vekt på at innsikt er noe som blir oppnådd med stødig og bevisst arbeid. Gestalt-modellen beskriver derimot innsikt som noe som kun kan komme etter at man først bevisst jobber så langt man kan med problemet, og når man da ikke kommer lenger, legger det til side for å la underbevisstheten jobbe videre med det. I ubevisstheten vil problemet mentalt omstruktureres, og det vil gi ny innsikt.

Å ha *kognitiv fleksibilitet* er noe som bidrar til å kunne omstrukturere problemer mentalt. Kognitiv fleksibilitet er evnen til å bytte mellom tankesett, strategier og oppgaver når man møter et hinder i problemløsning. På den andre side av kognitiv fleksibilitet finnes *kognitiv fiksering.* Kognitiv fiksering viser til å ikke klare å komme på kreative løsninger fordi man fikserer eller ikke vil gi opp på uproduktive strategier. Det skilles i litteraturen mellom to type fikseringer.

- Fiksering på en algoritme (Algorithmic fixation)
- Fiksering på overflateegenskapene til problemet (Content-universe fixation)

Å overkomme disse fikseringene er en viktig del av å være en vellykket problemløser i matematikk (Haavold & Sriraman, 2021).

Det er viktige forskjeller mellom noviser og eksperter innenfor matematikk når de løser matematiske problemer. Ekspertene bruker mer tid på å forstå problemet, planlegge og å lage representasjoner av problemet (Haavold & Sriraman, 2021). Noviser derimot bruker lite tid på å forstå problemet og planlegge. De lager sjeldent representasjoner av problemet.

Ekspertene går ofte fram og tilbake mellom alle de fire stegene i Polyas modell, avhengig av om strategien de valgte var vellykket eller ikke. Noviser derimot gjør ofte det første steget i Polyas modell kun en gang, selv om strategien de valgte ikke fungerte (Haavold & Sriraman, 2021).

Eksperter viser også større grad av kognitiv fleksibilitet når de arbeider med å løse problem. Dette kan bety å velge en ny strategi som er prinsipielt ulik den tidligere og mislykkede strategien. Noviser velger ofte nye strategier som er for like de tidligere, mislykkede strategiene. Dette betyr ikke at noviser ikke kan være kognitivt fleksible, men eksperter er oftere det (Haavold & Sriraman, 2021).

2.5 Lesters syn på problemløsning

Lester (2013) hevder at for å være en god problemløser i matematikk må problemløseren ha

- Erfaring i hvordan man lærer seg å løse problemer
- En stor kunnskapssamling i matematikk
- Ferdigheter i å bruke forskjellige representasjoner
- Evne til å kjenne igjen og lage inferensmønster

Problemløsning bør bli absorbert inn i et større begrep, noe Lester (2013) kaller *mathematical activity*. I *mathematical activity* har den metakognitive aktiviteten til individet eller gruppen stor betydning. Metakognitiv aktivitet i Lesters (2013) forståelse er en kontinuerlig regulering av sin egen problemløsing. Mathematical activity beskrives som en prosess med fire faser og begynner når en person står foran et komplekst problem.

I den første fasen må problemløseren forenkle kompleksiteten ved å identifisere konseptene og prosessene som har mest med problemet å gjøre. Dette medfører at problemløseren må velge hva som ved problembeskrivelsen kan ignoreres og hva som må bli tatt hensyn til. Ved å velge ut hva som er essensielt i definisjonen av problemet vil problemløseren utvikle en idé om hvordan de forskjellige konseptene er relatert til hverandre i problemet og da kunne lage realistiske representasjoner av problemet. Med forskjellige representasjoner av problemet, kan

da problemløseren utvikle en modell som er enklere å undersøke, manipulere og forstå enn den originale komplekse konteksten (Lester, 2013).

Deretter kommer abstraksjonsfasen. I abstraksjonsfasen introduseres matematiske konsepter og notasjoner for å representere de essensielle konseptene av problemet. Etter å ha introdusert de matematiske konseptene og notasjonene er problemet blitt til et spesifikt matematisk problem (Lester, 2013).

Når det originale problemet har blitt omformet til et matematisk problem skal det matematiske problemet løses. Da må problemløseren bruke sin base av matematiske fakta, ferdigheter og matematiske resonnement (Lester, 2013).

Siste fasen i mathematical activity innebærer å sammenligne resultatet fra det matematiske problemet med den originale konteksten. Lester (2013) påpeker at sammenligning kan foregå mellom alle de forskjellige fasene til enhver tid. Denne sammenligningen er en metakognitiv aktivitet som er essensiell i vellykket problemløsning (Lester, 2013).

2.6 Schoenfelds problemløsningsmodell

Schoenfeld (1992) har også undersøkt hvordan eksperter innenfor matematikk arbeidet med matematiske problemer og sammenlignet dette med hvordan nybegynnere i matematikk arbeidet med de samme problemene. For å undersøke dette ble problemløsningsprosessen delt opp i seks forskjellige aktiviteter. Denne modellen ble senere brukt for å kartlegge problemløsningsprosessen til elever og matematikere.

1. Lese
    - Dette er når problemløseren leser oppgaven eller referer direkte til tekst i problemet
2. Analysere
    - Denne delen av problemløsningen er karakterisert av at problemløseren prøver å fullt forstå problemet. Analyse-delen er velstrukturert og forholder seg til forutsetningene eller målene med problemet.
3. Utforske (Explore)
    - Utforskning er likt analyse, men mindre strukturert og man flytter seg til dels fra det originale problemet.
4. Planlegge
    - Dette innebærer å legge en plan for problemløsningen.

5. Implementere
    - Å implementere er å gjennomføre planen.
6. Verifisere
    - Å verifisere i problemløsningsprosessen er å vurdere løsningen man kom fram til.

I tillegg vurderer Schoenfeld (1992) når problemløseren gjør selvregulerende aktivitet, slik som å vurdere hvordan løsningsforsøket ser ut foreløpig. Schoenfeld (1992) hevder at å reflektere på problemløsningsprosess underveis har stor betydning for å være en effektiv problemløser. Effektive problemløsere har derfor høy grad av selvregulering underveis i problemløsningen.

Ifølge Schoenfeld (1992) kan mer enn halvparten av alle problemløsningsprosesser han har analysert modelleres som i Figure 1.

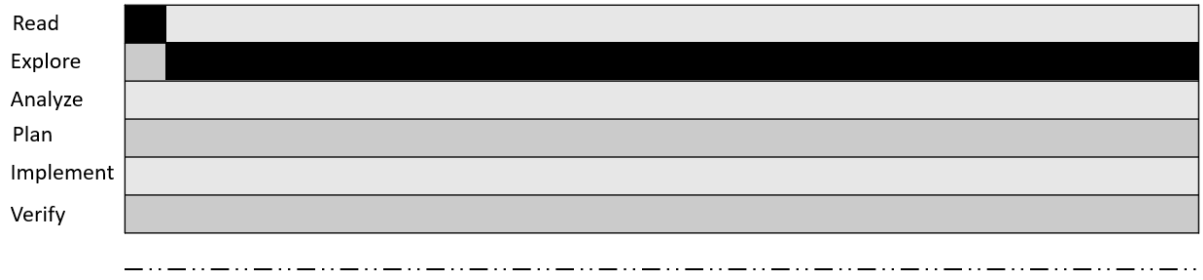


Figure 1:Problemløsningsprosessen til uerfarne problemløsere, ifølge Schoenfeld (1992)

I Figure 1 er de svarte feltene hvor problemløseren bruker tiden sin. Strekene på bunnen av figuren representerer tid. Det er ingen forskjell mellom de lysegrå og grå feltene, det er kun for å gjøre figuren lettere å lese.

I en slik prosess velger problemløseren en strategi eller et mål som problemløseren ønsker å bruke eller finne og bruker all tiden sin på det. Problemløsere som følger en slik prosess som i figuren kan karakteriseres som å ha liten grad av selvregulering i sin problemløsning.

2.7 Programmering

Hundrevis av forskjellige programmeringsspråk har blitt skapt siden de tidligste datamaskinene ble utviklet (O´Reagan, 2021). Programmeringsspråk deles av O´Reagan (2021) inn i fem distinkte generasjoner.

Et førstegenerasjons programmeringsspråk er et programmeringsspråk der man skriver inn i språket til datamaskinen. Slike programmeringsspråk består kun av 0-ere og 1-ere. Å skrive på datamaskinens eget språk betyr at programmet blir svært effektivt, da språket ikke må oversettes til språket til datamaskinen. En svakhet med slike programmeringsspråk var at de var svært vanskelige å lære seg og det var lett å gjøre feil, da det involverte å skrive en lang strøm av 0-ere og 1-ere. Det var heller ingen garanti at et program skrevet på denne måten ga samme resultat på en annen datamaskin. Førstegenerasjons programmeringsspråk ble hovedsakelig brukt på de tidligste datamaskinene.

Andregenerasjons programmeringsspråk var designet for en spesifikk datamaskin eller prosessor. Programmeringsspråket var da samme språk som prosessoren i datamaskinen anvendte. Det økte lesbarheten av programmeringsspråket, men medførte at prosessoren måtte konvertere språket til datamaskinspråket. Andregenerasjons programmeringsspråk har nesten blitt fullstendig erstattet av senere generasjons programmeringsspråk.

Tredjegenerasjons programmeringsspråk er laget for å anvendes for generelle applikasjoner, vitenskap og næringsvirksomhet. Eksempler på slike programmeringsspråk inkluderer C, Pascal og FORTRAN. Programmeringsspråkene hadde navngitte variabler, logiske operatorer og løkke-funksjoner blant annet. Disse var laget for å gjøre programmeringsspråkene enklere å forstå for mennesker, og var en stor fordel med tredjegenerasjonsspråkene. En annen fordel med disse programmeringsspråkene var at de var uavhengige av datamaskinen som de ble brukt på.

Fjerdegenerasjons programmeringsspråk er mer rettet mot hva som skal gjøres enn hvordan det gjøres. De er laget for å minske arbeidet med å programmere. Eksempler på slike programmeringsspråk er blant annet R og Mathematica. En ulempe med fjerdegenerasjons programmeringsspråk er at det opererer tregere enn tredjegenerasjons programmeringsspråk.

Femtegenerasjons programmeringsspråk er basert på å løse problemer ved å bruke begrensinger og forutsetninger i programmet, heller enn å bruke en algoritme lagd av programmereren. Målet er at datamaskinen skal løse problemet og ikke programmereren, og hvordan algoritme som brukes er ikke viktig i seg selv. Slike programmeringsspråk brukes blant annet for forskning på kunstig intelligens.

Det kan videre også skilles mellom tekst-basert programmering og blokk-basert programmering. Blokk-basert programmering har blitt brukt i stor grad de siste årene

(Weintrop & Wilensky, 2018). I blokk-baserte programmeringsspråk er den minste enheten en *node*. Det er i kontrast med tekst-baserte språk der de minste enhetene er tall, bokstaver eller tegn. Noder i blokk-basert programmering er designet på en måte der de settes sammen med hverandre, og hvis nodene er inkompatible kan ikke programmereren sette de sammen. Dette er for å forhindre syntaks-feil (Weintrop & Wilensky, 2018). Under er et eksempel på en node i det blokkbaserte programmeringsspråket Scratch.

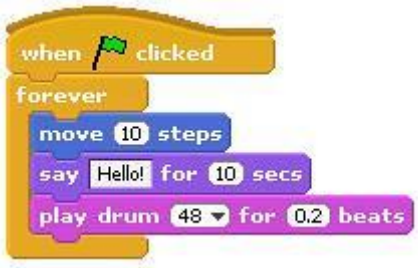


Figure 2: "To create Scratch programs, users snap together graphical blocks". From andresmh, CC BY-SA 2.0. https://www.flickr.com/photos/amonroy/2250306046

2.8 Algoritmisk tenking

I fagfornyelsen er programmering eksplisitt påpekt som en av de grunnleggende ferdighetene under digitale ferdigheter i matematikk (KD, 2020). Et annet begrep som brukes under kjerneelementet «Utforskning og Problemløsning» er begrepet *algoritmisk tenking.* Under vil begrepet algoritmisk tenking bli drøftet.

Algoritmisk tenking som brukt av Utdanningsdirektoratet (2019) er evnen til å dekomponere et komplekst problem til mindre, løsbare problemer. Det innebærer å analysere informasjonen i problembeskrivelsen med logikk, bruke det til å dele opp problemet til delproblemer, og så lage algoritmer for å løse delproblemene. En hovedmetode er å lage modeller og abstraksjoner av det komplekse problemet (fra den virkelige verden) ved å forenkle eller fjerne irrelevante detaljer i det komplekse problemet. Ved å løse delproblemene, kan det komplekse problemet bli løst. Løsningen på problemet eller løsningene på delproblemene kan deretter anvendes til å løse andre komplekse problemer eller delproblemer. *Algoritmisk tenking* er den norske oversettelsen av det engelske *computational thinking*.

Begrepet *computational thinking* ble for første gang brukt av Jeanette Wing (2006). Hun definerer å tenke algoritmisk som:

- Å løse problemer
- Å designe systemer

- Å forstå menneskelig oppførsel

Algoritmisk tenking bygger på de grunnleggende konseptene fra informatikken. Å kunne tenke algoritmisk er imidlertid noe annet enn å kunne programmere (Wing, 2006)

I kjølvannet av Wings arbeid har begrepet algoritmisk tenking blitt brukt mye. Forskere har laget mange forskjellige definisjoner for algoritmisk tenking. Shute, Sun og Clarke (2017) forsøkte å sammenligne disse definisjonene for å lage én felles definisjon av algoritmisk tenkning. Shute et.al (2017) deler oppfatningen til Wing (2006) om at algoritmisk tenking og programmering ikke er det samme. De hevder derimot at å ha kompetanse i algoritmisk tenking er en forutsetning for å kunne programmere. Definisjonen som de har laget, består av 6 deler.

- Dekomposisjon
- Abstraksjon
- Algoritmer
- Feilsøking
- Iterasjon
- Generalisering

Shute et.al (2017) definerer at å dekomponere et problem er å bryte problemet ned i mindre, løsbare delproblemer.

Abstraksjon har tre underdeler:

- Datainnsamling og analyse. Dette innebærer å samle inn de viktigste delene fra forskjellige datakilder.
- Mønstergjenkjenning. Dette handler om å gjenkjenne mønster i datasett.
- Modellering. Dette vil si å kunne bygge modeller eller simulasjoner for å vise hvordan et system fungerer eller hvordan systemet vil fungere framover i tid.

Algoritmer innenfor algoritmisk tenking betyr å lage instruksjoner som gjør at man kan finne løsningen til et problem. Innholdet av instruksene skal kunne utføres av et menneske eller en datamaskin. Algoritmer deles videre inn i fire underkategorier:

- Algoritme-design. Dette handler om å lage en rekke med instrukser for å løse et problem.

- Parallellisme. Dette vil si å kunne gjennomføre flere instrukser samtidig.
- Effektivitet. Dette betyr å kutte ned unødvendige steg, slik at algoritmen er så kort som mulig.
- Automasjon. Dette handler om å automatisere prosessen slik at man enkelt kan bruke algoritmen hvis man må løse relaterte problemer.

Feilsøking i algoritmisk tenking betyr å kunne finne og fjerne feil slik at løsningen blir korrekt.

Iterasjon vil si å gjøre flere omganger med designprosesser for å finne den optimale løsningen.

Generalisering betyr å kunne overføre ferdighetene fra algoritmisk tenking til andre områder for å løse problemer.

Den beste måten å lære seg algoritmisk tenking er gjennom programmering (Kazimoglu et.al, 2011, sitert i Psycharis & Kallia, 2017). Ifølge Einhorn (2012) bruker programmering alle elementene av algoritmisk tenking. Kunnskapen og erfaringene man får ved å løse programmeringsproblemer kan skape et rammeverk som man kan anvende i mange forskjellige fagfelt.

2.8.1 Algoritmisk resonnering

Algoritmisk resonnering (algorithmic reasoning) er et begrep som faller under det Lithner (2008) kaller *imitative reasoning.* Imitativ reasoning er karakterisert av å imitere en løsningsmetode fra en kilde. Lithner (2008) deler imitativ reasoning inn i to kategorier:

- Memorised reasoning
- Algorithmic reasoning

Strategien i memorised reasoning er å huske en korrekt løsning på et problem. Implementasjonen av strategien er deretter bare å skrive ned den korrekte løsningen. Et eksempel på memorised reasoning kan være å huske hvordan et matematisk bevis skal gjennomføres og deretter å skrive ned de nødvendige stegene.

I algoritmisk resonnering er strategien å huske en løsningsalgoritme for problemet. Å gjennomføre algoritmen som gir korrekt svar er trivielt for personen som gjennomfører den. Et kjennetegn med algoritmisk resonnering er at alle de vanskelige konseptuelle betraktingene

er håndtert av algoritmen. Lithner (2008) viser til at 7-åringer kan bli lært å derivere enkle polynomer uten å ha noe matematiske forståelse hvorfor resultatet blir som det blir. Imidlertid kan også personer som forstår hvorfor en løsningsalgoritme fungerer benytte seg av den.

2.9 Programmering i matematikkundervisning

Programmering ble innført i fagfornyelsen (KD, 2020), men det er ikke første gang programmering har blitt tatt inn i matematikkundervisninge. På 80-tallet ble programmeringsspråket LOGO svært populært i undervisning. Med LOGO skulle elevene styre en skilpadde på en dataskjerm, med å gi den forskjellige instrukser (Forsström & Kaufmann, 2018). Enkelte studier fant positiv effekt på elevers problemløsningsevne og sosiale ferdigheter når man brukte LOGO i matematikktimer, mens andre studier kunne ikke finne noe effekt. Simmons & Cope (1993) viste at bruk av LOGO for å løse problemer med vinkel og rotasjon bidro til at elevene holdt seg på et instrumentelt nivå, i motsetning til eleven som løste de samme problemene på papir. I senere tid har mengde med programmeringsspråk spesifikt laget for bruk i undervisning økt betydelig, som Scratch, LEGO Mindstorms og Micro:bit (Forsström & Kaufmann, 2018).

Ifølge Forsström og Kaufmann (2018) er det flere mulige gevinster med å bruke programmering i matematikkfaget, men det kreves mer forskning på området for å få bedre oversikt over relasjoner mellom programmering og matematikk. Å bruke programmering i matematikk kan øke elevenes motivasjon for å lære matematikk, da elevene får se matematikk brukt i en virkelig kontekst. Mye av programmeringen i matematikkfaget blir koblet opp mot geometri og viser en kobling mellom algoritmisk tenking og problemløsning innenfor geometri. Det trengs mer forskning på gevinstene med å bruke programmering innenfor andre temaer i matematikken, ifølge Forsström og Kaufmann.

En annet poeng er at matematikkundervisning med programmering ikke følger den tradisjonelle måten å undervise matematikk på. Elevene samarbeider i større grad og lærerens rolle blir forandret. En slik forandring er noe som er lagt vekt på i kjerneelementet «Utforskning og Problemløsning (KD, 2020).

2.9.1 Programmering i svensk skole

Sverige valgte i 2017 å integrere programmering i sin nye læreplan i matematikk. Kilhamn, Rolandsson & Bråting (2021) har sett på hvordan programmering ble brukt i

matematikktimene i Sverige. De skiller mellom fire forskjellige kategorier av bruk av programmering i matematikktimer:

A. Kun programmering
B. Matematikk som en kontekst for programmering
C. Programmering som verktøy for å gjøre beregninger
D. Programmering som verktøy for å utforske matematikk

I 33 % av timene handlet timen kun om programmering. I 38% av timene var matematikken kun en kontekst for å drive med programmering. Det vil si at hensikten med timen var å jobbe med programmering, men det var en matematisk kontekst til stede for å forsvare å programmere i en matematikktime. Omtrent 50 % av disse timene handlet om geometri i matematikk.

Dette kan forklares med at når programmering blir innlemmet i matematikkfaget, oppfatter lærerne at temaer innenfor programmering er en del av matematikkfaget. Deler av det kan også forklares med at programmering fortsatt var i ferd med å bli implementert i skolen når studien ble gjennomført. Videre hadde heller ikke lærere og elever mye erfaring med programmering fra tidligere trinn, noe som vil endre seg i framtiden.

Kategori C og D skiller seg fra kategori A og B da disse relaterer direkte til temaer innenfor matematikken.

I kategori C blir programmering sidestilt med andre matematiske metoder for å gjøre beregninger. Timene som ble kategorisert i kategori 3 var alle på høyere årstrinn, dette kan være fordi det er først der at beregningene som skal gjennomføres blir kompliserte nok for å forsvare bruken av programmering til beregning.

I kategori D blir programmering brukt for å utforske begreper og sammenhenger mellom begreper. Det argumenteres for at programmering ikke var den mest egnede måten å utforske matematikken på i timene som ble analysert. Andre digitale verktøyer kunne vært mer egnet ifølge studien.

I læreplanen i matematikk i Sverige skulle programmering knyttes spesielt opp mot algebra. Det viste seg at algebra var et tema der programmering ble brukt i liten grad. Begrepet variabel ble diskutert i mange av timene, men det var da variabel i programmering som ble diskutert i så fall.

Det er tydelig at i en stor del av matematikktimene der programmering brukes mangler matematisk innhold. Hvis det er slik at programmering i matematikkfaget kommer på bekostning av å undervise matematikk, kan det spørres om programmering egentlig burde integreres i matematikkfaget.

2.9.2 Effekter av å undervise programmering parallelt med matematikk

Psycharis & Kallia (2017) undersøkte effekter av å undervise programmering parallelt med matematikk. De ønsket å undersøke tre forskjellige effekter:

- Har det en effekt på elevers resonnement å undervise programmering parallelt med matematikk, og i så fall til hvilken grad?
- Har det en effekt på elevers mestringstro (self-efficacy) i matematikk å undervise programmering parallelt med matematikk, og i så fall til hvilken grad?
- Har det en effekt på elevers problemløsningsevne i matematikk å undervise programmering parallelt med matematikk, og i så fall til hvilken grad?

For å undersøke dette, ble det gjort flere tester på en kontrollgruppe og en eksperiment-gruppe. Kontrollgruppen hadde ikke programmering parallelt med matematikkundervisning, mens eksperiment-gruppen hadde det. Det Psycharis & Kallia (2017) fant var at det var en signifikant sammenheng i elevers evne til å resonnere, når de hadde programmering i lag med matematikkundervisning. Videre var det også en statistisk signifikant endring i elevers mestringstro for de som hadde hatt programmering sammen med matematikk.

Studiet fant derimot ikke en statistisk signifikant effekt på elevenes problemløsningsevne av å ha programmering og matematikk sammen. En mulig forklaring på dette kan være at elevene ikke ble trent i prinsippene for algoritmisk tenking, men i programmering. Psycharis & Kallia (2017, s. 15) sier “most students faced the problem rather as an exercise, than as a problem.”. Hadde elevene fått mer trening i algoritmisk tenking kunne de tydeligere ha sett stegene som skulle til for å løse problemet. En annen mulig forklaring kan være at elevene ikke fikk trening i generelle strategier innenfor programmering og matematikk. Det påpekes at videre forskning er nødvendig på dette området.

3 Metode

Innenfor forskning skiller man mellom kvantitative og kvalitative metoder. Den største forskjellen mellom disse metodene er hvordan man strukturerer undersøkelsen sin på forhånd. Det som menes med dette er hvordan forskeren vil samle inn og analysere datamaterialet på. I kvalitative metoder er ofte utvalget mindre og datainnsamlingen mer fleksible og åpen, da utvalgets egne perspektiver får større rom (Gleiss & Sæther, 2021).

Problemstillingen i denne oppgaven handler om hvordan en programmerer løser et matematisk problem. For å undersøke dette ble det matematiske problemet kjent som Josephus problem valgt. Dette problemet vil bli presentert i detalj lenger ned i teksten. I tillegg ville informanten bli stilt spørsmål etter å ha løst problemet.

3.1 Case-studie

Problemstillingen for denne oppgaven er som tidligere presentert å undersøke hvordan en programmerer løser matematiske problemer. For å undersøke problemstillingen ble det valgt å designe en case-studie. Case-studier er intensive studier som karakteriseres av fire aspekter, ifølge Andersen (2018):

- Undersøkelsen foregår i samtiden
- Undersøkelsen må ha rot i virkeligheten
- Fenomenet som undersøkes må ikke kunne skilles fra konteksten i det virkelige liv
- Undersøkelsen må ha mange og ulike datakilder

Case-studier er spesielt egnet å bruke når:

- Forskningsspørsmålet er et “hvorfor” eller “hvordan” spørsmål
- Forskeren kan ikke kontrollere hvordan undersøkelsen utvikler seg
- Man undersøker et samtidsfenomen, der det ikke er mulig å skille fenomenet fra konteksten i det virkelige liv

Problemstillingen i denne oppgaven inneholder et direkte “hvordan” spørsmål. Det andre punktet viser til at det er en problemstilling der man ikke kan analysere problemet med å variere parametere i problemet. Det er heller ikke mulig å ha en kontrollgruppe eller kontrolltest i denne problemstillingen. Videre er det er ikke mulig å gjennomføre undersøkelsen på nytt med nøyaktig samme betingelser.

3.2 Josephus Problem

Forskningsspørsmålet i denne oppgaven var «Hvordan løser en programmerer et matematisk problem?». Det ble valgt å bruke det matematiske problemet kjent som Josephus problem.

Josephus problem er oppkalt etter den jødiske historikeren som levde under det første århundre. Ifølge Josephus (u.å.) var han og hans menn fanget i en hule av romerne under beleiringen av Yodfat. Istedenfor å bli tatt til fange av romerne valgte de å ta livene sine. Josephus og en sammensvoren ønsket ikke å ta sine egne liv så Josephus foreslo at de trakk lodd og at den som fikk første loddet skulle bli drept av personen med det andre loddet. Slik skulle de fortsette til det var ingen igjen. Til slutt sto det bare igjen Josephus og hans sammensvorne, som overga seg til romerne.

Josephus problem i matematisk form har vært funnet i manuskripter fra det 10.århundre og i trykte bøker fra 16.århundre (Biggs, 1979). Problemet kommer i mange former, i noen varianter handler det om en båt der halvparten er kristne og andre halvparten er tyrkere. Halvparten av passasjerene på båten må hoppe over bord for å redde båten fra å synke. De bestemmer seg for å stille seg i sirkel, og at hver 15ende person må hoppe over bord. Problemet er da hvordan de kristne, eller tyrkerne, skal stille seg for å sikre at kun de er igjen i båten (Biggs, 1979).

Varianten av Josephus problem som ble brukt i intervjuet var formet på denne måten. Det er en ring med n antall krigere. Man velger en kriger og denne krigeren slår krigeren til venstre for seg. Deretter går man videre med klokken i ringen til man kommer til en levende kriger som deretter slår den nestelevende krigeren i rekken. Dette fortsetter til det kun står en kriger igjen. Gitt at man vet startpunktet, hvor ønsker man å stille seg for å være den siste krigeren som står igjen? Under følger tre figurer som illustrerer hvordan dette problemet ville gått hvis man hadde 6 krigere.

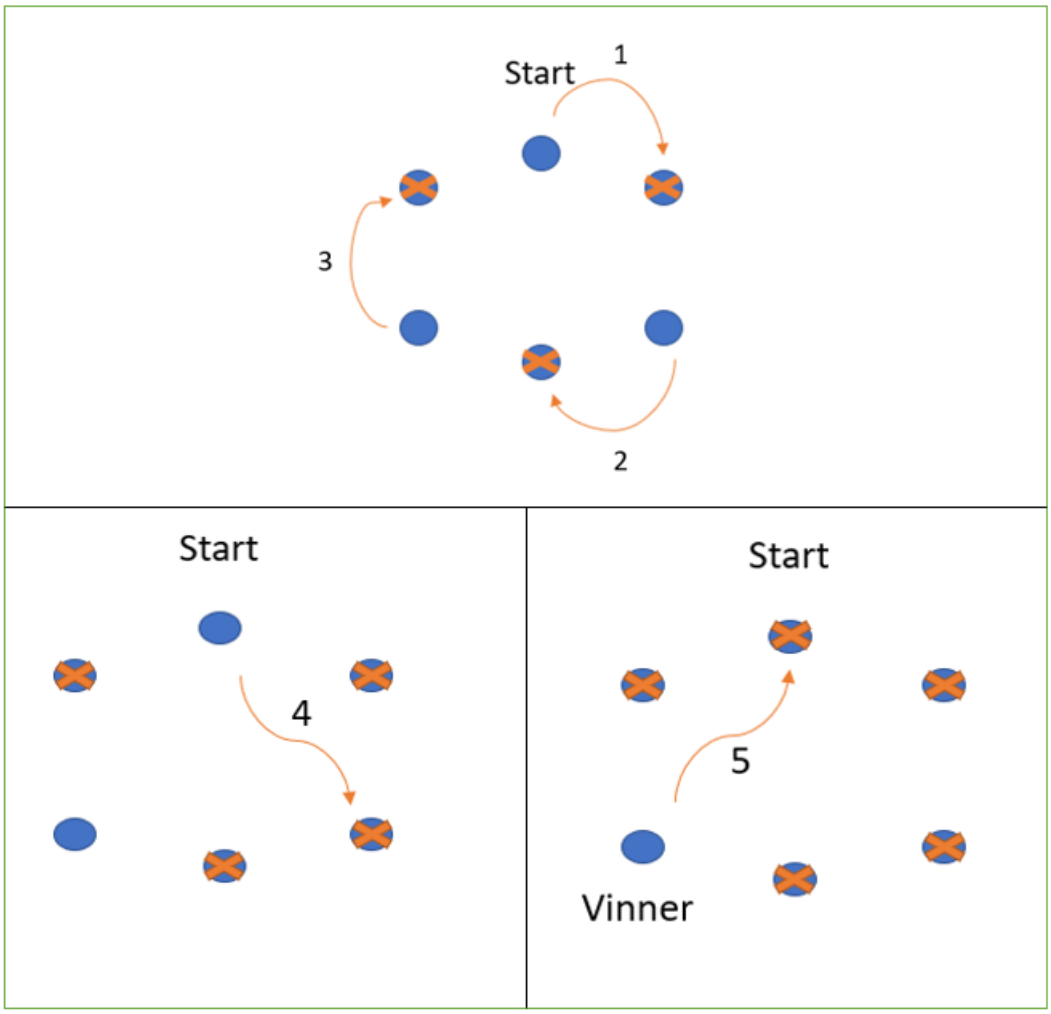


Figure 3: Josephus problem med 6 krigere

I Figure 3 er de blå prikkene krigere. Pilene viser hvilken kriger som slår en annen kriger, tallene over pilene viser hvilken rekkefølge de er i. De blå prikkene som har et kryss over er døde.

Øverst i Figure 3 slår den øverste krigeren (markert med Start over) krigeren til sin venstre, deretter slår den neste krigeren i rekken (som ikke er død) den neste krigeren som ikke er død. Hvis startposisjonen er posisjon 1, er det posisjonene 2, 4 og 6 som dør i den første runden rundt ringen.

Nede til venstre i Figure 3 slår krigeren i posisjon 1 krigeren i posisjon 3. Nede til høyre slår krigeren i posisjon 5 krigeren i posisjon 1. Da er det kun krigeren i posisjon 5 som står igjen i ringen.

3.3 Begrunnelse for valg av problemet

Hovedbegrunnelsen for å bruke dette problemet var at det er et matematisk problem som ikke har noen standardalgoritme for løsning. Det gjør at problemet kan oppfattes som et genuint problem for problemløseren, noe som er en forutsetning for å kunne undersøke hvordan problemløsning foregår. Hvis problemet hadde vært trivielt for informanten, er det usannsynlig at det hadde vært mulig å si noe av substans om problemløsningsprosessen til informanten. En annen grunn til å velge dette problemet er at det ikke krever stor kunnskap innenfor matematikk. Det gjør at de fleste vil kunne løse det.

En grunn til å inkludere dette problemet i intervjuet var at informanten skulle ha et eksempel å vise til under spørsmålene, som også intervjueren var kjent med. På denne måten kunne vi bruke tiden på diskusjonen og ikke å lete fram matematiske eksempler. Dette var også gunstig for å unngå misforståelser.

Hvorvidt dette matematiske problemet hadde noe med programmering å gjøre var interessant i seg selv. Her kunne informant og forsker ha forskjellige oppfatninger og dette kunne bli en interessant diskusjon.

3.4 Aktive samtalebaserte intervju

I tillegg til å la en programmerer forsøke å løse Josephus problem ble det valgt å stille spørsmål om programmering, matematikk og problemløsing etterpå. Intervju er en av hoved-metodene innen case-studier. Aktive samtalebaserte intervjuer preges av at forskeren bruker intervjuet til å vurdere sine egne oppfatninger samt forutsetningene til informantens observasjoner, beskrivelser og vurderinger. I slike intervjuer er begge partene aktive (Andersen, 2018). Man skiller gjerne mellom to type samtalebaserte intervjuer. I den ene typen er man interessert i informantens personlige opplevelser. I den andre typen er det interessant å intervju informanten fordi informanten er i en posisjon som gjør at de har innsikt i spesielle situasjoner, relasjoner eller tema som ikke andre har (Andersen, 2018). Det var den andre typen som var av interesse i vår undersøkelse. Slike intervjuer er egnet hvis informanten er ressurssterk (Andersen, 2018). Informanten som ble brukt er ekspert innenfor programmering, og siden dette er temaet som vi ønsket å diskutere kunne vi anta at informanten var ressurssterk innenfor dette.

3.5 Josephus problem i intervjuet

Formatet for hvordan problemløsingen skulle foregå hadde mange valg og vurderinger. Først ble problemet presentert muntlig av forskeren, samtidig som det ble tegnet opp et eksempel. Deretter fikk programmereren muligheten til å lese problemet formulert skriftlig. Dette var for å forsikre om at forskeren ikke hadde presentert problemet feil.

Programmereren fikk en skriveblokk og en penn utdelt. Det ble ikke presisert at hjelpemidler slik som kalkulator, internett eller lignende ikke var tillatt, men det ble ikke brukt underveis. Å tillatte hjelpemidler som for eksempel internett kunne gitt et annerledes og muligens feilaktig bilde på hvordan den matematiske problemløsingen hos en programmerer faktisk foregår.

3.6 Forskerens rolle

Hvilken rolle forskeren skulle ha en under problemløsingen hadde flere hensyn som måtte vurderes. Å være helt stille eller ute av rommet kunne muligens gitt et bedre innblikk i hvordan problemløseren ville oppført seg i sin mest naturlige form. En utfordring er da at programmereren antageligvis ikke ville formidlet underveis hvordan han tenkte gjennom problemløsningsprosessen. I tillegg er det en risiko for at problemløseren blir «stående fast» under problemløsingen, noe som kan ta lang tid eller gjøre at problemet ikke blir løst. At problemet ikke ble løst ville gi et ukomplett bilde av problemløsingen.

Ved å ha en mer aktiv rolle under problemløsingen var det mindre risiko for at programmereren ikke skulle fullføre problemløsingen, da det kunne bli gitt hint underveis eller diskutert hvordan problemløsingen går så langt. Det ble også mer naturlig for programmereren å formidle hva han tenkte underveis som kan gi gode innblikk i hvordan problemløsingen foregår. En svakhet med dette valget er at validiteten med problemløsingen senkes, da det ikke er en helt naturlig problemløsingssituasjon. Det var likevel denne metoden som ble valgt.

Når man skal gjøre intervjuer for å samle inn data, må man ta flere hensyn med tanke på forberedelse og gjennomføring av intervjuene.

Intervjuet skal være sosialt, men også være profesjonelt. Dette vil styre hva som blir diskutert i intervjuet og hva som ikke blir diskutert (Andersen, 2018). Det er derfor viktig å være

bevisst på dette, da dette blir en ny situasjon med annerledes roller enn vanlig for forskeren og informanten.

I forberedelsene til intervjuene har forkunnskapene til forskeren på det respektive området en stor rolle. I litteraturen er det ikke enighet om hvorvidt forskeren burde ha lite forkunnskaper på området eller mye forkunnskaper på området. Enkelte mener at forskeren bør ha så få forkunnskaper som mulig, da dette vil hindre at forskeren er forutinntatt (Andersen, 2018). Derimot kan mangel på forkunnskaper gjøre at forskeren ikke egentlig forstår hva som blir sagt i intervjuet (Andersen, 2018). Hva disse forkunnskapene baserer seg på er også av betydning. Hvis forkunnskapene til forskeren baserer seg i hovedsak på forskerens egne erfaringer, burde man trå varsomt. Det er viktig å ha faglig teori å støtte seg på.

Det er naturligvis ikke mulig å fjerne sine egne forkunnskaper før et intervju, selv om man så ville ønske dette. Det kan derimot være positivt at forskeren har forkunnskaper om programmering før intervjuene. Grunnen til dette er at programmering har særdeles mange begreper som er naturlig å bruke i en diskusjon om programmering. Eksempler på dette er variabel, funksjon, loop, klasse, liste, array etc. Hvis en ikke hadde forkunnskaper om disse begrepene ville noen av de vært uforståelige for forskeren og begreper som variabel og funksjon kunne blitt forvekslet med begreper fra matematikken med samme navn.

I selve gjennomføringen av intervjuet er det flere utfordringer. Det er viktig at intervjuet oppleves akseptabelt av informanten. En ting å unngå er å stille ledende spørsmål (Andersen, 2018). Å holde styringen på intervjuet kan også være en utfordring, da informanten har større kunnskap på feltet enn forskeren (Andersen, 2018).

3.7 Spørsmålene i intervjuet

Etter diskusjonen av Josephus problem ble følgende spørsmål diskutert.

1. Hvordan bruker du programmering i jobben din?
2. Hva vil du si det betyr å programmere?
3. Ser du noen likheter mellom å programmere og å løse en matematikkoppgave?
4. Tror du det er enklere for deg å løse en matematikkoppgave, siden du kan programmere?
5. Hva vil du si er de tre viktigste tingene vi har snakket om i dag?

Formålet med det første spørsmålet var å si noe kompetansen i programmering til informanten. Hvis informanten ikke hadde brukt programmering i jobben sin i det hele tatt, hadde nok ikke informanten vært gunstig for dette prosjektet.

Man kunne forvente at begrepet programmering kom til å bli brukt mye gjennom intervjuene. Det var derfor nødvending å spørre eksplisitt hva informanten definerer programmering som. Hvis dette ikke hadde blitt oppklart hadde nok forskerens egne forutsetninger om dette begrepet blitt definisjonen som ble brukt og det var da en risiko for å mistolke informantens utsagn.

Spørsmål 3. og 4. hadde som hensikt å komme inn på essensen av problemstillingen. Ved å stille spørsmål var det mulig at programmereren ville presentere sin måte å løse problemer innenfor programmering og matematikk. Dette kunne støtte opp hvordan problemløsingen foregikk under Josephus problem og gi dypere innsikt på programmererens problemløsingsmåte.

Det siste spørsmålet hadde som hensikt å få oppsummert det informanten selv mente var de viktigste momentene. Ved å stille dette spørsmålet ble det eksplisitt dokumentert hva som var de viktigste momentene fra informantens perspektiv.

I intervjuet ble en lydopptaker brukt for å dokumentere hva som ble sagt. Det er fordeler og ulemper med å bruke en lydopptaker. Lydopptaker gjør det mulig å gå tilbake å høre hva som faktisk ble sagt, dette er et gode man ikke får med å ta notater under intervjuet. Å bruke lydopptaker kan derimot også skape mindre frihet i intervjuet (Andersen, 2018).

3.7.1 Pilotering av intervjuet

Å være intervjuer var en ny og ukjent rolle. For å forsikre at spørsmålene fungerte i praksis og for å sikre seg at man ikke stilte ledende spørsmål, skulle intervjuet piloteres først. Intervjuet ble pilotert på en medstudent med bakgrunn i matematikk, programmering og matematikkdidaktikk.

Under piloteringen ble det oppdaget flere utfordringer. En utfordring var det å ikke stille ledende spørsmål, men samtidig stille spørsmål som holdt flyten i samtalen. Under pilotering kunne da gode spørsmål og problematiske spørsmål bli oppdaget.

Under diskusjonen av Josephus problem i starten av piloteringsintervjuet, ble intervjuobjektet «sittende fast» i en viss grad. Dette førte til en utfordring. I hvor stor grad det burde hintes

uten å røpe for mye var utfordrende. En konsekvens av dette var at det ble valgt å sette seg enda dypere inn i Josephus problem, da det ble lettere å bedømme hva som var akseptable hint.

Spørsmålet "Hva vil du si er de store forskjellene mellom programmering og å løse en matematikkoppgave?" viste seg utfordrende å svare på for intervjuobjektet under piloteringen. Dette kan tolkes på flere måter. Det kan tenkes at spørsmålet i seg selv var utformet på en måte som gjorde det vanskelig å svare på. Dette spørsmålet kom etter en lengre diskusjon om likhetene mellom programmering og å løse en matematikkoppgave. Intervjuobjektet hadde vært tydelig på at han synes det var betydelige likheter mellom å programmere og å løse matematikkoppgaver. Det er mulig at spørsmålet derfor ble vanskelig å besvare, siden intervjuobjektet ikke vil motsi seg selv. En annen tolkning er at spørsmålet i seg selv er vanskelig å besvare. Det kan også tolkes at intervjuobjektet ikke syntes det var noen forskjeller og det derfor ble vanskelig å svare.

Intervjuobjektet kontaktet meg etter intervjuet og ville diskutere spørsmålet ytterligere, han mente da at det var et interessant spørsmål som burde være med. Til intervjuet senere var det derfor viktig at det ble gitt tilstrekkelig tenketid på dette spørsmålet.

3.8 Informanten

I denne studien ble det valgt å bruke kun én informant. Informanten måtte oppfylle diverse kriterier for å kunne delta i prosjektet.

Det var nødvendig at informanten jobbet som programmerer og hadde gjort dette en stund. Dette var et naturlig valg, da en programmerer har den mest valide innsikten i hvordan programmerere løser matematiske problemer.

En annet kriterium var at informanten ikke skulle være utdannet matematiker. Dette var fordi at dette kunne gi et falskt bilde til hvordan programmere løser matematiske problemer. I en slik situasjon ville teknikkene og strategiene som ble lært under utdannelsen i matematikk kunne overskygge elementene fra problemløsingen som stammet fra å være programmerer.

Informanten som ble valgt ut vil bli referert til som «Per» i resten av teksten. Per er en mann i de senere 20-årene som jobber som programmerer og har gjort det i over 2 år. Per er utdannet fysiker og jobber med programmering innenfor fysikkfeltet. En betraktning med dette er at fysikk inneholder mye matematikk, som gjør at en fysiker arbeider mye med matematikk.

Dette kan medføre at en fysiker muligens vil bruke mange teknikker og strategier fra matematikken. Dette kan være en svakhet med informantvalget.

Å jobbe som programmerer kan være veldig forskjellig i forskjellige felt. Enkelte som jobber som programmerer jobber i hovedsak med programmering som verktøy for å beregne eller simulere noe. Andre programmerere jobber med å utvikle software, designe spill eller innenfor cybersikkerhet. Hvor tett matematikk er involvert i disse forskjellige rollene er nok varierende, men det er mulig at de som jobber med å beregne eller simulere har det nærmeste forholdet. Det vil derfor ha en betydning hvilken programmerer man intervjuer for hvilke svar man får på et intervju. Når det kommer til etterprøvbarhet til prosjektet, vil det derfor være nødvendig å intervjue noen fra omtrent samme felt.

Et sentralt spørsmål er hvorfor jeg valgte å kun bruke en informant. Det er flere grunner til dette. Hvis jeg skulle hatt flere informanter, måtte informantene hatt relativ lik bakgrunn. Grunnen til dette er at hvis informantene hatt forskjellig bakgrunn, alder, yrke, bosted etc., ville det vært utfordrende å sammenligne resultatene. Hvis man derimot hadde funnet flere informanter med relativt lik bakgrunn kan man spørre seg hvor mange informanter som blir "nok" (Andersen, 2018). En fordel med slike komparative studier er at man får en referanseramme på fenomenet man undersøker (Andersen, 2018). Tid og ressurser er naturligvis også en betraktning man må ta seg. Skal man gjøre to intervjuer på flere informanter blir datamaterialet også betydeligere større. Dette kan være en fordel, men det betyr at også at arbeidet blir betydeligere større.

Det er heller ingen garanti at et større datamateriale garanterer høyere kvalitet på datamaterialet. Hvordan man skal behandle et slikt datamateriale er også en betraktning man må ta. Med flere informanter er det naturlig å sammenligne hva som blir sagt i intervjuene, noe som ikke nødvendigvis er lett. Videre er det ikke uvanlig at det er høyt frafall i kvantitative studier.

3.9 Forskningsetikk

Det var viktig at prosjektet ble gjennomført på en etisk forsvarlig måte. For å sikre dette har flere steg blitt gjort.

Prosjektet har blitt vurdert og godkjent av NSD (Norsk senter for forskningsdata). I tråd med dette har informanten fått tilstrekkelig informasjon om prosjektet og sine rettigheter i et informasjonsskriv (se vedlegg). Per har samtykket til å delta og blitt informert om at han kan

trekkes seg når tid som helst. Videre er dataen anonymisert og skal slettes etter prosjektslutt. Dataen er også bare tilgjengelig til de involverte i prosjektet, det vil si student og veileder. Denne dataen ble lagret på en lokal datamaskin med passordbeskyttet mappe.

3.10 Analysemetode

I analysen av intervjuet, var det hensiktsmessig å se etter mønster i samtaleemnene. Det ble derfor innledende i analysen kategorisert og kodet. Dette vil hjelpe til å finne mønster (Postholm & Moen, 2020). Kodene som ble valgt ut var programmering, matematikk, algoritmisk tenking og problemløsning. Ved å kode transkripsjon ble det mulig å se om programmering ofte ble diskutert samtidig som problemløsning eller om dette ikke fant stedet.

I case-studier er det derimot ingen generelle modeller for å analysere data (Andersen, 2018). Man burde derimot gjøre klare forskning-strategiske valg for å forhindre at man gjør feilslutninger eller gjengir dataen feilaktig. For å sikre seg dette kan man se på tre faktorer når man gjør analysen. Man må gi en nøyaktig gjengivelse av fakta, i denne oppgaven vil det si å vise til direkte sitater ifra intervjuet. Deretter må man vurdere hva som kan være forskjellige forklaringer av fakta. Deretter må konklusjonen være den forklaringen som er mest i overenstemmelse med fakta.

Analysen av datamaterialet vil bruke flere forskjellige modeller for problemløsning, blant annet Polyas (Polya, 1957) modell, Gestalt-modellen (Haavold & Sriraman, 2021) og Schoenfelds (Schoenfeld, 1992) modell for kartlegging av problemløsning. Disse modellene vil bli brukt for å undersøke hvordan problemløsing av Josephus problem foregikk, men også for spørsmålene som kom etter Josephus problem.

3.11 Kvalitet

Kvaliteten på forskningen kan vurderes ut ifra dens *reliabilitet* og dens *validitet.* Reliabilitet måler kvaliteten på forskningsprosessen. Det sier noe om hvorvidt dataen har blitt formet av metodevalget. Videre er også reliabiliteten et mål på hvor reproduserbar dataen er (Gleiss & Sæther, 2021).

Validitet er et mål på selve datamaterialet og forskerens analyse og konkusjoner. Høy validitet krever at metoden og utvalget er velegnet for å svare på problemstillingen. I kvalitative studier er validitet et mål som sier noe om hvorvidt dataen reflekterer den faktiske verden (Gleiss & Sæther, 2021).

En svakhet med aktive samtalebaserte intervjuer, er at reliabiliteten er svakere enn ved inaktive intervjuer med faste, forhåndsbestemte spørsmål.

En styrke med intervjuer er at de har meget høy validitet. Intervjuer kan få fram de faktiske meningene til de involverte aktørene. Dette er noe som kan være betydelig vanskeligere å få til med for eksempel spørreundersøkelser (Andersen, 2018)

Under analysen er det ønskelig å prøve å bestemme i hvilken av Polyas (Polya, 1957) fire steg problemløsningen er til enhver tid. Det er derimot ingen garanti på at det ville gitt samme resultat for hver ekspert som utførte denne kartleggingen. Grunnen til dette er at stegene er generelle av natur, og hva en person mener er bevis for å lage en plan behøver ikke å samsvare med en annens oppfatning av samme situasjon.

I Gestalt-modellen legges det stor vekt på en mental restrukturering av problemet (Haavold & Sriraman, 2021). Å si at noe er bevis på en mental restrukturering er ikke lett, da det er umulig å ha innsyn i nøyaktig hvordan en person tenker. Videre kan det stilles spørsmål om når noe er restrukturert «nok» for å kalle det for en mental restrukturering.

Når det kommer til reliabilitet ved bruk av Schoenfelds modell (Schoenfeld, 1992), hevder Schoenfeld selv at forskjellige personer som bruker denne modellen for å kartlegge en problemløsningsprosess kan få samme resultat. Modellen er ikke uten problemer derimot, Schoenfeld (1992) peker på at for å få reliable resultater med bruk av denne modellen krever det at den som anvender har erfaring med å bruke denne modellen. I tilfellet med denne oppgaven hadde jeg aldri brukt modellen før. For å få erfaring med å bruke modellen har modellen blitt anvendt 3 ganger for problemløsningen av Josephus problem.

Videre er Schoenfelds (1992) analyse av problemløsning basert på at individene løser problemene alene, med minimal input fra andre. Dette er ikke tilfellet i denne masteroppgaven, noe som kan gjøre det vanskeligere å sammenligne resultatene.

En annen betraktning er hvorvidt resultatene av denne studien kan sammenlignes med resultatet av andre studier. Haavold & Sriraman (2021) har grupper av nybegynnere eller eksperter innenfor matematikk og analyserer hvordan de forskjellige gruppene løser problemer. I denne masteroppgaven er det kun en enkeltperson som løser problemer. Å sammenligne eksperter innenfor programmering med eksperter innenfor matematikk kan

derfor gi et annerledes bilde enn hvis det hadde vært en gruppe med eksperter innenfor programmering som løste problemet.

4 Data og analyse

Hvordan en programmerer går fram for å løse et matematisk problem var en stor del av intervjuet. Omtrent halvparten av intervjuet ble brukt til å diskutere Josephus problem. Denne prosessen skal sammenlignes med Polyas modell (Polya, 1957) og Gestalt-modellen (Haavold & Sriraman, 2021). Det blir også diskutert hvorvidt fiksering (Haavold & Sriraman, 2021) var til stede under problemløsningen. Videre vil også problemløsningsprosessen bli sammenlignet med Schoenfelds modell (Schoenfeld, 1992) for kartlegging av problemløsning.

Hvordan informanten selv mener problemløsing innenfor programmering og matematikk fungerer vil også bli diskutert.

4.1 Josephus problem

Forskjellen på hvordan eksperter og nybegynnere innenfor matematikk løser matematisk problemer er stor. Hvordan en programmeringsekspert løser et matematisk problem er interessant å undersøke da dette kan belyse hvorvidt en programmeringsekspert har mer til felles med nybegynnere i matematikk enn med matematikkeksperter. Det som gjør det interessant i en undervisningssammenheng er at programmering skal brukes i utdanningsløpet i større grad fremover og hvorvidt dette vil påvirke elevers matematiske problemløsningsevne. For å undersøke dette skal problemløsningsprosessen analyseres med Polyas modell (Polya, 1957) og Gestalt-modellen (Haavold & Sriraman, 2021). Hvor ofte det byttes strategi vil også bli undersøkt.

I starten ble problemet lagt fram muntlig for Per, og deretter leste Per oppgaven på papir. Etter dette stilte Per et oppklaringsspørsmål til forskeren «Det kan være helt tilfeldige mengde folk i den sirkelen?» (Per, Transkripsjon, 2022). Dette kan være for å forsikre som om at han har forstått problemet og at både han og jeg hadde lik forståelse av oppgaven. Dette er noe som ekspertene innenfor matematikk gjorde, da de alle leste oppgaven og deretter diskuterte oppgaven for å forsikre seg om at alle hadde lik forståelse av oppgaven (Haavold & Sriraman, 2021). Alternativt kan det være at Per stilte dette spørsmålet for at jeg skulle presisere hva som var forutsetningene for oppgaven, for å hjelpe han å forstå problemet. Denne forhåndsdiskusjon var rimelig kort, omtrent 1 minutt. Dette kan være på grunn av naturen av problemet, at Per følte at problemet var forstått eller at dette ikke føltes nødvendig. En annen bemerkning er at ekspertene innenfor matematikk løste oppgavene i grupper, mens Per løste dette problemet alene.

Etter den korte diskusjonen, lagde Per en konfigurasjon med 16 krigere. Denne konfigurasjonen ble tegnet ned på papir og løst. Det virker ikke som at Per brukte noe tid på å legge en plan på dette stadiet, men gikk fram med en tilsynelatende tilfeldig konfigurasjon. Det er også mulig at Per hadde lagt en plan, men uten å formidle dette muntlig til meg. Etter denne konfigurasjonen ble løst dannet Per seg en hypotese, «Kanskje man vil være han første?» (Per, Transkripsjon, 2022), men bemerker også «det var sikkert for at det var 16 da.» (Per, Transkripsjon, 2022).

Deretter ble det laget en ny konfigurasjon med 9 krigere og følgende bemerkning ble også gjort «Hmm, det er forskjellig svar tror jeg på partall og oddetall kanskje?» (Per, Transkripsjon, 2022). Denne bemerkningen ble gjort før problemet med 9 krigere ble løst. Etter å ha funnet svaret på konfigurasjonen med 9 krigere sa Per følgende «Ok så på 9 så var det da nummer ikke nummer han første, men nummer 3 da. På 16 så ville det være han første og det vil det mest sannsynlig være på alle andre også. Så hvis vi prøve 8.» (Per, Transkripsjon, 2022). Med alle andre mente Per «Alle andre partallsløsninger det var tanken» (Per, Transkripsjon, 2022). Per lagde da en konfigurasjon med 8 krigere og løste denne, og hadde følgende å si om dette «Det går på 8 og det går på 16, så da tror jeg egentlig at vi gambler på at det er en sannhet i det her og siden når det er partall så halveres det med 2, så tar du modulo 2 av det så ender du på 0. Jeg har en følelse at det har noe å si her.» (Per, Transkripsjon, 2022). Dette er en feilaktig konklusjon, men en struktur i problemløsningen som er tydelig. Per hadde en hypotese og lagde deretter en plan for å teste dette og gjennomførte planen. Deretter brukte Per resultatet for å vurdere sin hypotese og begrunnet resultatet med en forklaring. Dette har flere likhetstegn med Polyas modell (1957).

Etter å ha funnet et resultat for partallene, sa Per følgende «Det blir ikke noe rest av alle partallsløsningene. Og på når det var 9, så var det nummer 3. 3, 4, ok på alle så er det sånn, hvis vi da prøver et annet partall, typ 11.» (Per, Transkripsjon, 2022). Jeg spurte da «Annet partall?» (Birk, Transkripsjon, 2022). Per sa «Nei annet oddetall, typ 11.». Dette kan tolkes som at Per hadde funnet et tilfredsstillende svar for partallene og ville sette i gang med å finne et svar for oddetallene. Dette er dekomposisjon som definert av Shute et.al (2017), som er et kjennetegn på algoritmisk tenking. Per lagde da en konfigurasjon med 11 krigere, og satt i gang med å løse denne. Etter Per løste denne konfigurasjonen sa han «Hmm på oddetall så er det litt vanskelig å se med en gang.» (Per, Transkripsjon, 2022). Dette var det første tidspunktet der Per sto fast.

Etter en liten stund med stillhet, sa jeg «Du mener at på alle partallene så vil det alltid være gunstig å være han som starta.» (Birk, Transkripsjon, 2022). Per sa da «Ja, det er det jeg har kommet fram til nå ihvertfall, nei oi jeg har ikke sjekka, jeg har kun sjekka 4 gangen vet du, 4, 8, 16, 32. Ah farsken. Vi prøver 6 her da.» (Per, Transkripsjon, 2022). Det kan tolkes som at Per gjør en vurdering av hvordan han fant ut av dette resultatet, da han betviler om det er korrekt. En annen forklaring kan også være at jeg brakte fram dette resultatet og han gjetter på at det må være feil i så fall. Per lagde deretter en konfigurasjon med 6 krigere og verifiserte at man ikke alltid være i startposisjonen når det er partallskonfigurasjoner.

4.1.1 Problemet i et runde-tankesett

Etter verifikasjonen ble Per stående fast igjen. Jeg spurte da «Så la oss si at, her har du jo 16 posisjoner. I den første runden når man går rundt, hvem er som dør da?» (Birk, Transkripsjon, 2022). Per svarte da «Da er det alle partallene som dør.» (Per, Transkripsjon, 2022). Jeg spurte «Hva er neste runde da?» (Birk, Transkripsjon, 2022) og Per svarte «Da hvis man har 16, så er det alle annethvert oddetall da typ, så neste runde så dør nummer 3, 7, 11, 15.» (Per, Transkripsjon, 2022) og Per fulgte opp med «Da er det 3, 7, 11, 15 og runden etter der, så dør da 5, 13 og så dør nummer 9 til slutt kanskje?» (Per, Transkripsjon, 2022). Per forklarte dette mønsteret med «Så for antall runder da, så dem dør alltid I partall. Eller altså fra runden starter så er det alltid partallene av de overlevende som dør.» (Per, Transkripsjon, 2022).

Dette var en annerledes måte å betrakte problemet på, da hver runde ble analysert individuelt. En slik mental restrukturering kan forklares med Gestalt-modellen, da det å møte et logisk stopp kan føre til en mental restrukturering av problemet (Haavold & Sriraman, 2021). En alternativ forklaring kan også være at jeg stilte spørsmål hva som skjedde i hver runde og da Per ble med på denne mentale restrukturering av den grunn. Uansett hva årsaken til den mentale restruktureringen var viser det at Per har kapasitet til å sette seg inn i alternative tankesett, noe som er et tegn på kognitiv fleksibilitet (Haavold & Sriraman, 2021). Med en slik måte å betrakte problemet på dekomponeres problemet til å løse hver runde. Dekomposisjon av problemer er et kjennetegn ved algoritmisk tenking (Shute et.al, 2017).

Etter Per løste konfigurasjonen med 16 krigere, undersøkte han om mønsteret ville være det samme for en konfigurasjon med 11 krigere. Per konkluderte med «Ja hmm, da blir det samme mønster som nummer 16, ok nå skal jeg teste. Da virker det som at når det er partall så lønner det seg å være han som starter runden. Når det er partallene. Men for hvert oddetall, hver oddetallsrunde så kanskje det lønner seg å være han som er hakket før.» (Per,

Transkripsjon, 2022). I sitatet til Per kommer det fram at han ønsker å teste noe spesifikt, noe som kan tyde på at en plan har blitt lagd, og det settes i gang med å gjennomføre planen. Dette er i tråd med Polyas problemløsningssteg (Polya, 1957). En annen bemerkning er at Per viser til et resultat om at det lønner seg å starte i partallsringer, noe som han selv beviste var usant. Dette kan forklares med en fiksering (Haavold & Sriraman, 2021) eller simpelthen med at han glemt resultatet.

4.1.2 Problemet i et rest-tankesett

Under løsningen av konfigurasjonen med 11 krigere sa Per «Så, han nummer 1 starte, drep 2, nummer 3 tar 4, 5 tar 6, 7 tar 8, 9 tar 10, og 11 han står da igjen, så da kan vi si at runden er ferdig med 1 i rest.» (Per, Transkripsjon, 2022). Her introduserer Per et nytt begrep som han kaller for rest. Begrepet rest blir brukt i sammenheng med runder. Det kan tolkes som en mental restrukturering av problemet (Haavold & Sriraman, 2021). Det kan også tolkes som at dette ikke er en komplett restrukturering av problemet, da begrepet rest blir brukt i sammenheng med runde-tankesettet. Å innføre et nytt begrep for å forklare hva som skjer i hver runde kan tolkes som å lage en representasjon av problemet. Dette er noe ekspertene i matematikk gjorde i betydelig større grad enn nybegynnere (Haavold & Sriraman, 2021). Å bytte tankesett er et tegn på kognitiv fleksibilitet (Haavold & Sriraman, 2021).

Videre under løsningen av konfigurasjonen med 11 krigere sa Per «Så starter neste runde og da er dem 6 stykker, ... Da er det partallsrunde så da i teorien min er det lønnsomt å være han, men det er det jo ikke.» (Per, Transkripsjon, 2022). Per holder fortsatt fast på det feilaktige resultatet om at det lønner seg å være i startposisjonen i en partallskonfigurasjon. Det kan argumenteres at dette er en fiksering (Haavold & Sriraman, 2021).

Per begynte deretter å regne ut den neste runden i konfigurasjonen med 11 krigere. «… så i neste runde så tar han nummer 11 og får da drep nummer 1, nummer 3 får drep nummer 7, nummer 7 får drep nummer 9. Og nummer 11 er da nok en gang en rest. Så nå har han vært rest 2 ganger på rad. Hmm, han vært rest 2 gang, så får nummer 11 drep nummer 3, så får nummer 7 drep nummer 11. Hmm» (Per, Transkripsjon, 2022). Per påpeker her at posisjon har vært rest 2 ganger på rad. Det kan tolkes som at Per vurderer resultatene underveis i problemløsningen. Det er i tråd med det Schoenfeld (1992) hevdet matematikere gjorde under problemløsning.

Etter å ha løst konfigurasjonen med 11 krigere sier Per «Da var det nummer 7 som lønt seg. Ja. Hmm. 11 stykker posisjon 7 lønnet seg. 4 unna 11. … han som sto foran var rest to gang. Kanskje...» (Per, Transkripsjon, 2022). Det kan tolkes som at Per tror at det kan være en sammenheng med antall rest i en runde og vinnerposisjonen. Det virker også som et eksempel der resultatet blir vurdert og analysert, det er bruk av det siste steget i Polyas modell (Polya, 1957).

Under løsningen av denne spesifikke konfigurasjonen med 11 krigere kan det argumenteres at Per har vært innom alle av Polyas steg (Polya, 1957). Å forstå problemet i Polyas øyne er å undersøke forutsetningene og begrensingene av problemet (Polya, 1957). Per løste en partallskonfigurasjon med 16 krigere og brukte resultatet av det for å danne en hypotese om oddetallskonfigurasjon, nemlig at i oddetallsrunder vil vinnerposisjon i den runden være 1 posisjon tidligere enn startposisjonen. Etter å ha dannet hypotesen, lages det en plan for å undersøke hypotesen. Planen hans var å lage en konfigurasjon med 11 krigere. Å lage en plan er det andre steget i Polyas prosess (Polya, 1957). Deretter gjennomfører Per de nødvendige stegene for å løse konfigurasjon. Å gjennomføre planen sin er Polyas tredje steg (Polya, 1957). Til slutt vurderes og analyseres resultatet, som er det siste steget i Polyas problemløsningsprosess (Polya, 1957). Å bruke alle stegene i Polyas prosess er noe eksperter innenfor matematikk gjør i større grad enn nybegynnere i matematikk (Haavold & Sriraman, 2021).

Per kommenterte ikke noe mer på hypotesen «Men for hvert oddetall, hver oddetallsrunde så kanskje det lønner seg å være han som er hakket før» (Per, Transkripsjon, 2022), men lagde en ny konfigurasjon med 5 krigere. Per sa «Ok så da prøve vi en ny med 5. Så tar vi et annet oddetall, 11 er et primtall kanskje det ødelegger ting med regelen.» (Per, Transkripsjon, 2022) idet han lagde konfigurasjonen. Ideen om at primtall kunne ha en betydning for løsningen kom uten at det hadde vært noe diskusjon om primtall tidligere. Å se problemet fra nye sider er et tegn på kognitiv fleksibilitet (Haavold & Sriraman, 2021). Jeg kommenterte «5 er og et primtall.» (Birk, Transkripsjon, 2022). Etter denne kommentaren ble ikke primtallsteorien fulgt videre.

Per løste konfigurasjonen med 5 krigere og la umiddelbart en plan deretter «Ok så her var det kun en runde med rest og da lønnet seg å være nummer 3. Og så når det var 11 var det to runder med rest og da lønt seg å være nummer 7. Så hvis vi får til en runde med, la oss prøve med 19 da.» (Per, Transkripsjon, 2022). Valget av å lage en konfigurasjon med 19 krigere ble

forklart med «Tre runda med rest for da å se om det da lønne seg å være nummer 11.» (Per, Transkripsjon, 2022). Det kan tolkes som at resultatet av konfigurasjon med 5 krigere var basisen for hva neste konfigurasjon skulle være. Resultatet vurderes og analyseres for å forstå problemet, det er i tråd med Polya (1957).

Per oppdaget underveis mens han gjennomførte regningen for konfigurasjon med 19 krigere at teorien hans ikke stemte. Det kommenterer han selv i dette sitatet «Og så tar nummer 7 nummer 11, der røk den teorien.» (Per, Transkripsjon, 2022). Per gjennomfører likevel hele utregningen for konfigurasjonen med 19 krigere, og påpeker følgende til slutt «… for da ta nummer 15, det var da nummer 7 som leve igjen. Var ikke det det samme som på 11?» (Per, Transkripsjon, 2022).

Per sa dette om resultatet «Grunnen til det ... Vi starta da med 19 og vi starta da med 11 og samme person som overlevd. Kan det være med at mellom 11 og 19 er det helt clean partall, hmm kanskje ikke, det er 8 ... hvor er den med 11 tro?» (Per, Transkripsjon, 2022). Per danner fort en hypotese om at det at det er et «clean»-partall kan ha en betydning for resultatet. Det er uklart hva Per mener med «clean partall», hypotesen forkastes også etter noen sekunder med tenketid. Etter å ha forkastet hypotesen ønsker Per å se på konfigurasjon med 11 krigere. Det kan tolkes som at han ønsker å sammenligne disse konfigurasjonene. Da bruker han resultatet av en konfigurasjon for å forstå problemet og for å lage en plan, slik som i Polyas prosess (Polya, 1957).

Etter å ha funnet konfigurasjonen med 11 krigere, sa Per «11 så var det nummer 7 som overlevd, og der var han 2 runder med rest. Her så var han 2 runder med rest og 1 med rest og det hadde ikke noe å si siden det var fortsatt nummer 7 her.» (Per, Transkripsjon, 2022). Det kan tolkes som at Per konkluderer med at rest i runder ikke har noe betydning for hvilken posisjon som vil være lønnsom, da han sier «… det hadde ikke noe å si …» (Per, Transkripsjon, 2022). Å vurdere resultater på denne måten er likt med Polyas siste steg (Polya, 1957).

4.1.3 Problemet brutt ned i delproblemer

Etter Per hadde sammenlignet de to konfigurasjonene og vurdert de to resultatene sa jeg «I første runden så fjerne man alle partallene, så når vi var i 19 så fjerne vi 2, 4, 6, 8, 10, 12, 16, 18 så vi reduserer problemet til et problem, siden ...» (Birk, Transkripsjon, 2022). Per sa da «10 deltakere da, siden vi fjerner 9 stykker» (Per, Transkripsjon, 2022). Jeg sa da «Men det

blir kanskje litt feil å si redusere problemet, siden vi ender jo opp med et annet startpunkt.» (Birk, Transkripsjon, 2022). Per sa da «Ja det blir jo et nytt problem typ. 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 da. Ok nå prøver vi å se på det som at det blir et nytt problem for hver gang.» (Per, Transkripsjon, 2022).

Å behandle problemet som at det blir et nytt problem for hver gang var en ny strategi. Å bruke nye strategier for å løse problemet viser kognitiv fleksibilitet (Haavold & Sriraman, 2021). Hvor denne nye strategien kom fra, har flere mulige forklaringer. En forklaring kan være at det stammer fra mitt sitat «… vi reduserer problemet til et problem …» (Birk, Transkripsjon, 2022). En annen forklaring kan være at denne strategien ble lagd ubevisst som følge av at Per var kommet til et stopp i problemløsningen, noe som ville vært i tråd med Gestalt-modellen (Haavold & Sriraman, 2021). Tidligere kom den mentale restrukturering der Per så på problemet som runder med rest etter jeg hadde en kommet med en lengre kommentar. De lengre kommentarene mine kom som regel etter Per hadde fått et stopp i problemløsingen, dette gjør det vanskelig å vite hva som var kilden til den mentale restrukturering som Per gjorde nå, da jeg igjen kom med en lengre kommentar etter et stopp i problemløsningen.

Planen var å behandle problemet som at det ble et nytt problem mellom hver runde. Per iverksatte denne planen med å lage en konfigurasjon med 15 krigere. Under løsningen av delproblemet med 15 krigere hadde Per dette å si «Her har vi 15 stykker, 1 starter, 2 ryk, 4 ryk, 6 ryk, 8, 10, 12, 14. Og da får vi plutselig et nytt problem og da er hvor mange i ringen? 2 stykker det var han som starta, 3, 4, 5, 6, 7, 8, 9. Gir det mening tro? 1, 2, 3, 4, 5, 6, 7, 8 nei det er bare 8 stykker» (Per, Transkripsjon, 2022). Pers kommentar «Gir det mening tro?» (Per, Transkripsjon, 2022) kan tolkes som selvregulering (Schoenfeld, 1992), da Per vurderer resultatet underveis om det er rimelig.

Nå hadde altså problemet med 15 krigere blitt til et problem med 8 krigere. Per sa «Ok så da har vi gått fram 15 til 8 og vi har mista 7 og da er vi ned i et partallsproblem.» (Per, Transkripsjon, 2022). Strategien med å behandle problemet som at det ble et nytt problem ble fortsatt helt til Per løste den siste konfigurasjonen.

Per la fram en hypotese etter å ha løst det siste delproblemet i problemet med 15 krigere. Per sa «At med en gang man havner i en 2-gange er det den som står på topp som overlevde, men det er bare hvis at du kan dele det helt ned. Altså hvis du kan dele det på 2 helt ned til 1.»

(Per, Transkripsjon, 2022). Det kan argumenteres at hele prosessen er i tråd med Polyas (Polya, 1957) problemløsningsprosess da Per forstår problemet i en kontekst, legger en plan for å løse problemet, gjennomfører planen og vurderer og bruker resultatet til å forstå problemet ytterligere. En slik prosess ble brukt mer av eksperter innenfor matematikk enn nybegynnere i matematikk (Haavold & Sriraman, 2021).

For å bruke dette resultatet sa Per følgende «Jeg var inne på tanken med å primtallsfaktorisere på antall folk i ringen for å fjerne alle 2er elementene i tallet. Men så tenker jeg jo at 11 og 19 har vi jo prøvd på og de hadde samme svar på 7. Begge dem er primtall da.» (Per, Transkripsjon, 2022). Å primtallsfaktorisere antallet krigere i ringen var en ny og annerledes strategi som ble foreslått, den var annerledes fra de andre da denne strategien ikke ville kreve å tegne opp konfigurasjoner. Strategien ble derimot ikke anvendt videre, det kan likevel tolkes som kognitiv fleksibilitet (Haavold & Sriraman, 2021).

Per påpekte «Så hvis det er en 2er potens så lønner det seg alltid å starte. Den tror jeg at vi kan sett spikra.» (Per, Transkripsjon, 2022). Etter dette resultatet sa Per «Siden når vi var 19, 2er potenser på det er, da kommer du til 16 og så har du 3 i rest. Men på 19 så var det 7 som var tallet. 19, 3 i rest, 7, 11, 3 i rest, 7, her hadde jeg 15, 7 i rest, og da endte vi opp på nummer på nummer 15 som ble seirende.» (Per, Transkripsjon, 2022). Per la deretter en plan, der han ønsket å lage en konfigurasjon der det ville bli 5 i rest etter å ha delt tallet i den høyeste 2er potensen pluss rest. Dette kommer fram fra «Ok, da vil jeg teste en som har 5 i rest. 13 typ. Siden da virker det jo som at antall rest fra start på den første delinga, den indikerer hvor du bør stå hen i sirkelen.» (Per, Transkripsjon, 2022). Det kan tolkes at en slik prosess er likt med Polyas (Polya, 1957) problemløsningsmodell.

Dette viste seg å være en vellykket strategi. Etter å ha lagd en konfigurasjon med 13 krigere og løst denne konfigurasjonen sa Per «… så du vil være hvis du tar startallet og så finn du resten av det og så tar du og ganger det med 2 pluss 1 så er det posisjon du vil ha.» (Per, Transkripsjon, 2022). Etterpå ga Per en mer presis formulering av løsningen, «… posisjon blir lik rest ganger 2 pluss 1» (Per, Transkripsjon, 2022). Til slutt skrev Per denne løsningen ned som en formel på papiret.

Gjennom hele problemløsningsprosessen viser Per stor grad av kognitiv fleksibilitet (Haavold & Sriraman, 2021), da mange forskjellige strategier blir anvendt gjennom problemløsningen. Noen av strategiene eller tankesettene som blir brukt er så forskjellige at det kan tolkes som

en mental restrukturering (Haavold & Sriraman, 2021). Dette er i tråd med hva som skilte eksperter i matematikk fra nybegynnere i matematikk under problemløsning (Haavold & Sriraman, 2021).

De mentale restruktureringene som forekom, skjedde som regel etter et stopp i problemløsningen. Dette er likt med Gestalt-modellen (Haavold & Sriraman, 2021). En bemerkning angående dette er at de mentale restruktureringene oppsto etter dialog med forskeren. Dette gjør det usikkert om de mentale restruktureringene kom som følge av at det ble et stopp i problemløsningen og Per da omstrukturerte problemet underbevisst eller om dialogen spilte større rolle.

Strategiene som ble brukt ble som regel produsert av en prosess som er lik Polyas modell (Polya, 1957). Per beveger seg gjennom de fire stegene i Polyas modell på en sirkulær måte da Per prøver å forstå problemet, lager en plan, utfører planen og vurderer resultatet. Resultatet blir deretter som regel brukt for å forstå problemet ytterligere og da lage en ny plan. Et slik sirkel-mønster var noe som eksperter innenfor matematikk anvendte i større grad enn nybegynnere i matematikk (Haavold & Sriraman, 2021).

I problemløsningen var det også tegn på fiksering (Haavold & Sriraman, 2021). Det var i hovedsak en fiksering på feilaktige resultater. Et feilaktig resultat var spesielt motstandsdyktig da Per holdt fast på dette resultatet selv etter å ha selv sagt at det ikke stemte.

4.1.4 Analyse av Josephus problem med Schoenfelds modell

I dette kapitlet vil Pers løsning av Josephus problem bli kartlagt med Schoenfelds modell (Schoenfeld, 1992). Schoenfelds modell inneholder seks forskjellige aktiviteter, disse er 1) lesing, 2) utforskning (explore), 3) analysering, 4) planlegging, 5) implementering og 6) verifisering. Dette produserte følgende figur.

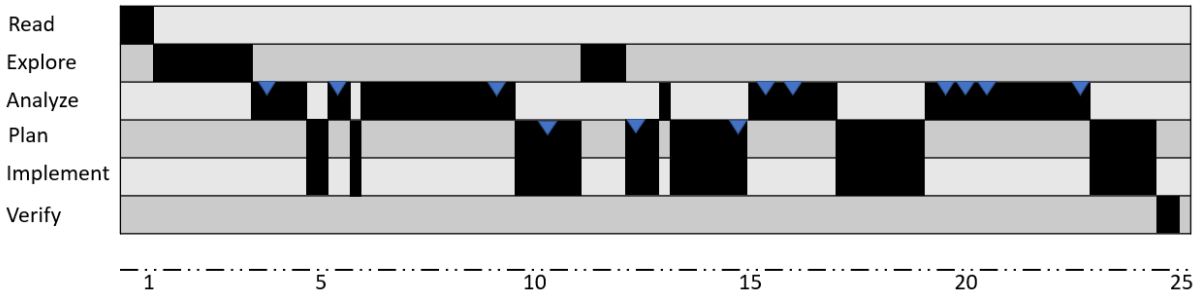


Figure 4: Problemløsningsprosessen til Per

Aksen nederst markerer antall minutter. Trekantene representerer hver gang Per gjorde eller sa noe som kan tolkes som selvregulering. Et eksempel på dette er «… jeg har kun sjekka 4 gangen vet du, 4, 8, 16, 32. Ah farsken. Vi prøver 6 her da.» (Per, Transkripsjon, 2022). Her vurderer Per hvordan problemløsningen går så langt og reflekterer over hvilke partallskonfigurasjoner han har prøvd å løse. Per bestemte seg da for å sjekke en annen partallskonfigurasjon. Trekantene representer også selvregulering som førte til uproduktive strategier eller feilaktige resultater.

Plan og implementering aktivitetene er ofte markert som at de foregår samtidig. Det er to grunner til dette. Den ene grunnen er at forklaringen av planen foregikk som ofte samtidig som Per implementere planen. Den andre grunnen er at tidsenhetene er små.

Tid brukt i hver fase er representert i tabellen under.

Fase	**Tid brukt**
Read	45 sekunder
Explore	3 minutter 15 sekunder
Analyze	10 minutter 45 sekunder
Plan & Implement	8 minutter og 15 sekunder
Verify	30 sekunder
SUM	24 minutter 30 sekunder

Figure 5: Tidsbruken til Per i Schoenfelds (1992) modell

Nesten all tiden i problemløsningsprosessen er brukt i de to fasene *analyze* og *plan and implement*. Matematikere brukte en betydelig større mengde tid i analyse fasen (Schoenfeld, 1992).

Uerfarne problemløsere har som tendens å bruke nesten all tid i *explore* fasen (Schoenfeld, 1992). Ut ifra figur 4 er det tydelig at dette ikke er tilfellet med Per. Det kan derfor argumenteres at Per ikke er en uerfaren problemløser.

Det brukes også en betydelig mengde tid i *plan* og *implement* fasen. Det kan ha flere forklaringer. Det kan skyldes naturen av problemet av problemet selv. En annen forklaring

kunne vært at dette er Pers særegne stil på å løse problemer. Å legge mange forskjellige planer og gjennomføre dem kan være en konsekvens av å *dekomponere* (Shute et.al, 2017) problemet ned i delproblemer. Å dekomponere problemet er et kjennetegn på algoritmisk tenking (Shute et.al, 2017). Å programmere er en av de mest effektive måtene å trene algoritmisk tenking på (Einhorn, 2012) så det kan argumenteres at en programmerer som Per ville være god på algoritmisk tenking.

Effektive problemløsere er ofte selvregulerte (Schoenfeld, 1992). Fra figur 4 ser vi at det er mange tilfeller hvo Per viser selvregulering. Det kan derfor argumenteres at Per er en effektiv problemløser. Det er derimot ikke mulig å si hva som er kilden til selvreguleringen med denne modellen.

Det er mange tegn på at Per er en effektiv problemløser i matematikk. Dette kan muligens forklares med at Per er kompetent innenfor algoritmisk tenking. Programmering har vist seg å være en aktivitet som fostrer algoritmisk tenking (Einhorn, 2012), og det kan derfor argumenteres at Per er god på algoritmisk tenking. Å være god på algoritmisk tenking innebærer å kunne bruke teknikker, strategier og konsepter fra informatikken for å løse problemer i andre fagfelt (Wing, 2006).

4.2 Spørsmålene i intervjuet

Etter Per hadde løst Josephus problem fulgte en diskusjon om likhetene og forskjellene mellom programmering og matematikk. Disse spørsmålene vil bli diskutert nedenfor. For å forstå hva Per mener med begrepet programmering vil dette belyses først.

4.2.1 Pers bruk av begrepet programmering

Det var forventet at begrepet programmering kom til å bli brukt hyppig under intervjuet. Det var derfor ønskelig å få innsikt i hva dette begrepet betydde for Per. Per sier dette om programmering, «… programmering er jo bare implementering av algoritmer» (Per, Transkripsjon». Senere i intervjuet ble det også sagt «Begge [matematikk og programmering] er jo teknikker for å problemløse» (Per, Transkripsjon, 2022). Det kan tolkes at for Per er programmering er en teknikk for å problemløse, der teknikken innebærer å implementere algoritmer.

Ifølge Per innebærer det å kunne programmere at en blir «moderne-type handy» (Per, Transkripsjon, 2022). Å kunne programmere handler ifølge Per hvordan problemer i jobb- og skolesammenheng takles, dette kommer fra sitatet «Fordi både på hvordan man da kan takle

problemer som man støter på, i hvert fall i skole og jobbsammenheng. Dem problemene som vi ofte har. Så er det at veien til å finne en løsning på det er ikke nødvendigvis så lang at, siden man kan jo programmere, så vet man litt hvordan man skal sette det opp og det betyr at man kan takle ganske ulike komplekse problemer» (Per, Transkripsjon, 2022). Jobb- og skolesammenheng i denne konteksten er generell, og det er uklart om Per mener sammenhenger som ikke har noe med informatikk å gjøre. Hvis det gjelder andre felt enn informatikken er dette et syn som har likhetstrekk med algoritmisk tenking som definert av Wing (2006).

4.2.2 Likheter mellom problemløsing i programmering og matematikk

Å løse en matematikkoppgave har flere steg ifølge Per, «… man ser problemet i tekstformat som regel. Ut ifra fra det vet man hvordan type problem man har, og da vet man hvilken ligning man skal sette opp for å løse det, og så får man et svar når man er ferdig.» (Per, Transkripsjon, 2022). Ifølge Per har det å programmere noen felles likhetstrekk med å løse matematikkoppgaver, «I hvert fall på måten man setter opp oppgaven, vil jeg si det er likt med programmering, på at når du skriver koden som skal regne det så setter du opp oppgaven i det språket. Begge er jo teknikker for å problemløse.» (Per, Transkripsjon, 2022).

Ifølge Per er måten oppgaven «settes opp» på helt lik i matematikk og programmering, dette er tydelig fra sitatet «Den biten i hodet når du skal sette opp oppgaven, fra du leser problemet til du faktisk skriver det ned på papiret, eller skriver det inn i programmeringsspråket. Siden prosessen tror jeg er helt lik, uavhengig om du gjør det med matte eller programmering. Fordi det er rett og slett å tolke problemet.» (Per, Transkripsjon, 2022). En slik tilnærming for å løse problemer der det er et stort fokus på å tolke problemet før man går i gang med å skrive noe deler likhetstrekk med å forstå problemet i Polyas (Polya, 1957) modell for problemløsning. I Pers oppsett innebærer det å tolke problemet før man skriver, mens i Polyas modell (Polya, 1957) er det å forstå problemet før man legger en plan.

Denne «oppsettbiten» i programmering har mange elementer ifølge Per. Det innebærer først «… hvordan du vil strukturere koden din» (Per, Transkripsjon, 2022). Videre i oppsettbiten «hvis du har et problem, så må du liksom sette det opp sånn at du har en input til problemet i koden» (Per, Transkripsjon, 2022). Deretter følger å «lager en funksjon som tar imot en input og så behandler du den inputen på, dem logiske reglene som du tror trengs for å løse problemet.» (Per, Transkripsjon, 2022). Til sist i oppsettbiten følger «Og så kan du se på

outputen og samsvare det med det du trodde, og da definere om du, hvis du vet at funksjonen din er riktig så har du fått riktig svar.» (Per, Transkripsjon, 2022).

Det kan tolkes som at oppsettbiten er karakterisert av følgende steg.

1. Planlegge kodestruktur
2. Lage input
3. Lage funksjon som behandler input
4. Verifisere output

Det første steget handler om å legge en plan på hvordan koden burde struktureres, her er planlegging i fokus. I de to neste stegene er det implementasjon av input og funksjoner som er målet. Til sist følger verifikasjon. En slik prosess har denne strukturen 1) planlegge, 2) implementere, 3) verifisere.

Å løse problemer i matematikk eller i programmering har en helt lik prosess ifølge Per. Fra dette sitatet belyses denne prosessen, «fordi selve prosessen er jo helt lik. Les oppgaven, analysere, sette opp, få et svar.» (Per, Transkripsjon, 2022). Ifølge Per er det fire steg i å løse et problem i matematikk og programmering

1. Les oppgaven
2. Analysere
3. Sette opp
4. Få et svar

Å sette opp tolkes som å være det samme som oppsettbiten i programmering. Da blir problemløsning i matematikk og programmering å følge en slik struktur.

- Les oppgaven
- Analysere
- Sette opp
    - Planlegge
    - Implementere
    - Verifisere
- Få et svar

Hva steget «analysere» består av og betyr er ikke eksplisitt forklart av Per. Siden dette steget finner sted før problemet blir satt opp, kan det tyde på at det er mer overordnet arbeid med problemet. Hvis dette er tilfellet kan det tolkes som likt det første steget i Polyas modell (Polya, 1957) som går ut på å forstå problemet. I tillegg er neste steg å sette opp problemet, noe som virker vanskelig å gjøre uten å ha forstått problemet. Det kan argumenteres for at «analysere» steget derfor har som formål å forstå forutsetningene og begrensningene av problemet.

I neste del av problemløsningsprosessen følger det å sette opp problemet. I første omgang planlegges den overordnede kodestrukturen som er ønskelig for å løse problemet. Deretter implementeres planen, med å lage variabler og funksjoner. Til slutt må resultatet verifiseres. Det kan argumenteres for at disse stegene er lik steg 2, 3 og 4 i Polyas modell (Polya, 1957). Steg 2 i Polyas modell er å lage en plan, noe som er likt med Pers prosess. Deretter følger det å gjennomføre planen sin, noe som er likt i både Polyas modell og Pers modell. Til sist i Polyas modell er det å verifisere resultatet og å se tilbake på problemet. I Pers modell er det påpekt at resultatet skal verifiseres og sammenlignes med det du forventet. Forskjellen i dette steget fra Polyas og Pers modell er hvor mye vekt som er lagt på å reflektere over resultatet. I Polyas modell er dette et viktig element av å se tilbake på problemet.

Etter å ha satt opp problemet gjenstår steget «få et svar» i Pers modell. Under oppsett steget ble det påpekt at verifikasjon var en del av steget «sette opp», så steget «få et svar» handler ikke om å verifisere resultatet. Det kan tolkes som et steget «få et svar» innebærer å bruke det verifiserte resultatet til å få et svar på problemet. En mulig forklaring for dette steget kan være at problemer i programmering kan gi resultater i form av et tall, tabell, graf, simulering, etc., og at dette resultatet deretter må tolkes og anvendes for å få et svar på problemet.

Polyas modell (Polya, 1957) og Pers modeller har mange likheter, og det kan argumenteres for at de er mer lik enn ulik, der den største ulikheten er refleksjon på problemet når problemet er løst.

4.2.3 Ulikheter i problemløsing i programmering og matematikk

Per påpekte at problemløsningsprosessen i programmering og matematikk er helt lik. Per mener derimot at det er enkelte elementer med problemløsning som er forskjellige i programmering og matematikk.

Et slikt eksempel handler om visualisering av problemet, «hvis du får en vanskelig matteoppgave, så er det første du gjør er jo å sette deg ned å tegne. Og visualisere det.» (Per, Transkripsjon, 2022). I programmering derimot sier Per «I programmering er det ikke nødvendigvis det. Det første du gjør er bare du begynner å skrive på en liten kode også prøver du å tenke deg fram til det. Du er kanskje ikke like fokusert på å visualisere problemet i form av figurer, når man programmerer.» (Per, Transkripsjon, 2022).

Å visualisere problemet i form av figurer eller slikt kan også brukes i programmering og Per sier «som regel prøver jeg å programmere det først, men hvis det er et vanskelig problem, så begynner jeg til slutt å tegne på papiret. Og da viser det seg jo at litt enkel matte på papiret, kan være den starten du trenger for å få det til. Å få visualisert det.» (Per, Transkripsjon, 2022). Å tegne på papir er altså ikke noe unikt med å løse problemer i matematikk, men å tegne er noe Per oftere gjør hvis det er et matematisk problem. Ulikheten er altså i hvor stor grad denne teknikken blir anvendt i de forskjellige domenene.

En annen ulikhet handler om udefinerte variabler. Per sier «en annen ulikhet er vel kanskje at i programmering så er det kanskje mye vanskeligere å regne med udefinerte variabler. I programmering må du gjerne ha definert opp variablene dine til tall.» (Per, Transkripsjon, 2022). Det kan tolkes som at ikke alle problemer er egnet for å løses med programmering. En forskjell mellom problemløsning i matematikk og programmering er altså samlingen av problemer som kan løses med programmering eller matematikk ifølge Per.

Å løse en matematikkoppgave for hånd har noe eget med seg ifølge Per. Per sier «Når man gjør matematikk for hånd så blir du jo, du havne jo typ litt nærmere oppgaven da … Det er et lite skille mellom det å ha det på skjermen og å ha det på papiret foran seg» (Per, Transkripsjon, 2022). Det er uklart hva nøyaktig som menes med «havne nærmere oppgaven». En mulig forklaring kan være det å visualisere problemet som Per pekte på tidligere. En annen betraktning er at å løse matematikk for hånd så kan mulighetene for å lage representasjoner av problemet være større, da det kan være enklere å lage prøvefigurer uten å være begrenset av å ha tilgjengelig programvare. En annen forklaring kan være hvor mye fokus det er på de overordnede egenskapene til problemet. Det er mulig at det er et større fokus på dette i matematikk, enn for et tilsvarende problem i programmering.

For å forstå hvor likt problemløsing i programmering og matematikk ifølge Per, gir dette sitatet under en diskusjon om forskjeller i programmering og matematikk et godt innblikk.

«… derfor slet jeg også litt med ulikhet, siden det blir litt mer abstrakte ulikheter at følelsen er annerledes når du gjør det, fordi selve prosessen er jo helt lik. Les oppgaven, analysere, sette opp, få et svar.» (Per, Transkripsjon, 2022)

4.3 Konklusjon

I denne oppgaven var målet å undersøke hvordan en programmerer arbeider med matematiske problemer.

Under problemløsingen av Josepshus problem viser programmereren stor grad av kognitiv fleksibilitet (Haavold & Sriraman, 2021), noe som er også noe eksperter innenfor matematikk viser under problemløsing. Programmereren bruker Polyas fire steg (Polya, 1957) på en sirkulær måte, som også eksperter innenfor matematikk gjør. Selvregulering under problemløsing er noe som kjennetegner effektive problemløsere (Schoenfeld, 1992), og programmereren viste høy grad av selvregulering under problemløsingen. Det var også tegn på fiksering (Haavold & Sriraman, 2021) under problemløsingen.

Programmereren mener selv at den generelle prosessen, lese oppgaven – analysere – sette opp – få et svar, er helt lik i programmering og matematikk. Det er enkelte nyanser som er ulikt for problemløsing i matematikk og programmering, slik som at å visualisere problemet i form av figurer er mer brukt i problemløsing i matematikk.

I og med at en programmerer er en effektiv problemløser i matematikk kan det argumenteres for at programmering kan fostre gode problemløsningsvaner i matematikk. Hvis dette stemmer betyr det at elever kan være godt tjent med å ha både programmering og problemløsing som sentrale temaer i matematikkundervisningen.

Da programmereren løste Josephus problem ble det gitt input fra forskeren, dette kan skape et feil bilde på hvordan en programmerer løser matematiske problemer individuelt. Det er mulig at å la programmereren løse problemet uten input fra forskeren ville gitt et mer presist bilde på hvordan problemløsingen fungerer. Derimot ville det vært betydeligere vanskeligere å få innblikk i hvordan programmereren tenkte underveis i problemløsingen.

En bemerkning er at i denne oppgaven er det brukt mye tid på å sammenligne hvordan en gruppe med matematikkeksperter løste problemer i grupper og hvordan en programmerer løste et problem. Hvilken effekt gruppedynamikken har på problemløsingen er ikke åpenbar, noe som kan senke validiteten på sammenligningen.

4.4 Fremtidig arbeid

En programmerer og eksperter innenfor matematikk har mange likhetstrekk når de løser matematiske problemer. En fremtidig komparativ studie der programmerere og eksperter innenfor matematikk løser samme matematiske problem kan gi ytterligere innblikk i hvor likt problemløsingen er, og hva som eventuelt er forskjeller.

Videre kunne en case-studie om programmeres problemløsningsmetode der det ikke blir gitt input fra forskeren underveis muligens gi flere svar på hvordan problemløsingen foregår.

Det finnes programmerere i mange forskjellige felt. Cybersikkerhet, software, fysikk er eksempler på dette. Det er ikke gitt at programmerere fra forskjellige felt løser matematiske problemer likt, dette kunne vært interessant å undersøke.

Referanseliste

Andersen, S. (2018). Casestudier (2. Utg). *Fagbokforlaget*

Biggs, N, L (1979). The roots of combinatorics. *Historica Mathematica*, 6(2) 109-136

Einhorn, S (2012). Micro-worlds, computational thinking and 21st century learning. *White Paper: Logo Computer Systems Inc*. http://www.microworlds.com/support/files/lcsi-computational-thinking.pdf

Forsström, S. E. & Kaufmann, O. T. (2018). A Literature Review Exploring the use of Programming in Mathematics Education. *International Journal of Learning, Teaching and Educational*, 17(12).

Gleiss, M & Sæther, E. (2021). Forskningsmetode for lærerstudenter (1. Utg). *Cappelen Damm Akademisk.*

Haavold, P. Ø. & Sriraman, B. (2021). Creativity in problem solving: integrating two different views of insight. *ZDM – Mathematics Education.* https://doi.org/10.1007/s11858-021-01304-8

Jospehus, Flavius (u.å.). The works of Flavius Josepus: in three volumes; with illustrations. Oversatt av William Whiston. *George Routledge & Sons*. https://archive.org/details/worksofflaviusjo1873jose/page/578/mode/2up

Kilhamn, C., Rolandsson, L. & Bråting, K. (2021). Programmering i svensk skolmatematikk. *LUMAT: International Journal on Math, Science and Technology Education. University of Helsinki*

https://journals.helsinki.fi/lumat/article/view/1457/1503

Kirke- og undervisningsdepartementet (1987). Mønsterplanen for grunnskolen: M87. *Aschehoug*. https://www.nb.no/nbsok/nb/2aef891325a059851965d5b8ac193de5#0

Kunnskapsdepartementet (2020). *Læreplan i matematikk 1. – 10. trinn (MAT01-05).* Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2020. https://www.udir.no/lk20/mat01-05/om-faget/grunnleggende-ferdigheter?lang=nob

Kunnskapsdepartementet (2020). *Kjerneelementer*. Fastsatt som forskrift. Læreplanverket for Kunnskapsløftet 2020. https://www.udir.no/lk20/mat01-05/om-faget/kjerneelementer

Lester, F (2013). Thoughts About Research On Mathematical Problem-Solving Instruction. *The Mathematics Enthusiast.* 10 (1-2), 245-278.

Lithner, J (2008). A research framework for creative and imitative reasoning. *Educational Studies in Mathematics.* 67, 255-276.

O´Reagan, G (2021). History of Programming Languages. *A Brief History of Computing* 177-200. https://link-springer-com.mime.uit.no/chapter/10.1007/978-3-030-66599-9_15

Polya, G. (1957). How to solve it (2. Utg). *Princeton University Press*.

Postholm, M & Moen, T (2020). Forsknings- og utviklingsarbeid i skolen. En metodebok for lærere, studenter og forskere (2. Utg). *Universitetsforlaget*

Psycharis, S & Kallia, M (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving skills. *Instructional science* 45 (5) 583-602.

Root-Bernstein, R. (2003). Problem Generation and Innovation. *International Handbook on Innovation*. https://www.researchgate.net/publication/241783200_Problem_Generation_and_Innovation

Schoenfeld, A. H. (1992). Learning to think mathematically: Problemsolving, metacognition, and sense-making in mathmatics. I D. Grouws (Ed.), *Handbook for Research on Mathematics*

Schoenfeld, A. H. (1992). On Paradigms and Methods: What do you do when the ones you know don't do what you want them to? Issues in the analysis of data in the form of videotapes. *The journal of learning science* 2 (2) 179-214. https://www.jstor.org/stable/1466838

Shute, V. J. Sun, C. Clarke, J (2017). Demystifying computational thinking. *Educational Research Review* 22(1). https://www.sciencedirect.com/science/article/pii/S1747938X17300350?via%3Dihub

Shute, V. J. (1991). Who is likely to acquire programming skills? *Journal of Educational Computing Research*, 7(1), 1-24. https://myweb.fsu.edu/vshute/publications.html

Simmons, M. and Cope, P. (1993). Angle and Rotation: Effects of Different Types of Feedback on the Quality of Response. *Educational Studies in Mathematics 24*, 163-176.

Utdanningsdirektoratet (2019). Algoritmisk tenking. https://www.udir.no/kvalitet-og-kompetanse/profesjonsfaglig-digital-kompetanse/algoritmisk-tenkning/

Wing, J. M (2006). Computational thinking. *Communications of the ACM* 49 (3) 33-35. https://dl.acm.org/doi/10.1145/1118178.1118215

Weintrop, D & Wilensky, U (2018). Comparing Block-based and Text-based Programming in High School Computer Science Classsrooms. *ACM Transactions on Computing Education* 18 (1) 1-25 https://dl-acm-org.mime.uit.no/doi/abs/10.1145/3089799

Vedlegg

Vedlegg 1: Informasjonsskriv

Vil du delta i forskningsprosjektet

Samspillet mellom programmering og problemløsing i matematikkundervisningen fra en programmerers perspektiv

Dette er et spørsmål til deg om å delta i et forskningsprosjekt hvor formålet er å undersøke hvilken rolle programmering kan ha i å gjøre elever til bedre problemløsere i matematikkfaget. I dette skrivet gir vi deg informasjon om målene for prosjektet og hva deltakelse vil innebære for deg.

Formål
I denne mastergradsoppgaven vil jeg se på hvilken rolle programmering kan ha i å gjøre elever til bedre problemløsere i matematikkfaget. Det innebærer å se på problemløsning innen matematikk og innenfor programmering og hva som er likhetene og forskjellene på dette.

Hvem er ansvarlig for forskningsprosjektet?
UiT – Norges Arktiske Universitet – Instituttet for matematikk og statistikk er ansvarlig for prosjektet.

Hvorfor får du spørsmål om å delta?
Du blir spurt om å delta i dette prosjektet da du jobber i et yrke, der programmering brukes i arbeidshverdagen. Det er kun deg som har blitt spurt i å delta i dette forskningsprosjektet.

Hva innebærer det for deg å delta?
Hvis du velger å delta i dette prosjektet vil det innebære at du deltar på to intervjuer. Det første intervjuet vil vare rundt 30 minutter, mens det andre intervjuet vil være et kortere oppfølgingsintervju på rundt 15 minutter.

Intervjuet vil inneholde spørsmål om hva det innebærer å programmere, hva problemløsing innenfor matematikk/programmering innebærer samt eventuelle forskjeller på dette.

Det er frivillig å delta
Det er frivillig å delta i prosjektet. Hvis du velger å delta, kan du når som helst trekke samtykket tilbake uten å oppgi noen grunn. Alle dine personopplysninger vil da bli slettet. Det vil ikke ha noen negative konsekvenser for deg hvis du ikke vil delta eller senere velger å trekke deg.

Ditt personvern – hvordan vi oppbevarer og bruker dine opplysninger
Vi vil bare bruke opplysningene om deg til formålene vi har fortalt om i dette skrivet. Vi behandler opplysningene konfidensielt og i samsvar med personvernregelverket.

Personene som vil ha tilgang på dine opplysninger er student (Birk Eirikssønn Heiberg) og veileder (Anne Birgitte Fyhn).

Dine personopplysninger vil blitt tatt opp på en ekstern lydopptaker og lydopptakene vil være lagret lokalt på PC, med passordbeskytting på filen.

I oppgaven vil verken yrke, bosted eller alder være spesifisert direkte. Andre personopplysninger som kan komme fram i intervjuet vil bli redigert eller slettet.

Hva skjer med opplysningene dine når vi avslutter forskningsprosjektet?

Opplysningene anonymiseres når prosjektet avsluttes/oppgaven er godkjent, noe som etter planen er 1. juni 2022. Ved prosjektslutt vil datamaterialet anonymiseres.

Dine rettigheter
Så lenge du kan identifiseres i datamaterialet, har du rett til:

- innsyn i hvilke personopplysninger som er registrert om deg, og å få utlevert en kopi av opplysningene,
- å få rettet personopplysninger om deg,
- å få slettet personopplysninger om deg, og
- å sende klage til Datatilsynet om behandlingen av dine personopplysninger.

Hva gir oss rett til å behandle personopplysninger om deg?
Vi behandler opplysninger om deg basert på ditt samtykke.

På oppdrag fra UiT – Norges arktiske universitet, instituttet for matematikk og statistikk har NSD – Norsk senter for forskningsdata AS vurdert at behandlingen av personopplysninger i dette prosjektet er i samsvar med personvernregelverket.

Hvor kan jeg finne ut mer?
Hvis du har spørsmål til studien, eller ønsker å benytte deg av dine rettigheter, ta kontakt med:

- UiT – Norges arktiske universitet, Instituttet for matematikk og statistikk ved Anne Birgitte Fyhn *(anne.fyhn@uit.no og* 77660243) og Birk Eirikssønn Heiberg (*birk.heiberg11@gmail.com og +4747344640).*
- Vårt personvernombud: Joakim Bakkevold *(personvernombud@uit.no og 776 46 322).*

Hvis du har spørsmål knyttet til NSD sin vurdering av prosjektet, kan du ta kontakt med:

- NSD – Norsk senter for forskningsdata AS på epost (personverntjenester@nsd.no) eller på telefon: 55 58 21 17.

Med vennlig hilsen

Anne Birgitte Fyhn og Birk Eirikssønn Heiberg

(Forsker/veileder) (Student)

Samtykkeerklæring

Jeg har mottatt og forstått informasjon om prosjektet "Samspillet mellom programmering og problemløsing i matematikkundervisningen fra en programmerers perspektiv", og har fått anledning til å stille spørsmål. Jeg samtykker til:

☐ å delta i to intervjuer

Jeg samtykker til at mine opplysninger behandles frem til prosjektet er avsluttet

---

(Signert av prosjektdeltaker, dato)